

THESIS / THÈSE

MASTER EN SCIENCES INFORMATIQUES

Implémentation d'un système de gestion de fichiers à structure arborescente

Majérus, Patrick

Award date:
1983

Awarding institution:
Université de Namur

[Link to publication](#)

General rights

Copyright and moral rights for the publications made accessible in the public portal are retained by the authors and/or other copyright owners and it is a condition of accessing publications that users recognise and abide by the legal requirements associated with these rights.

- Users may download and print one copy of any publication from the public portal for the purpose of private study or research.
- You may not further distribute the material or use it for any profit-making activity or commercial gain
- You may freely distribute the URL identifying the publication in the public portal ?

Take down policy

If you believe that this document breaches copyright please contact us providing details, and we will remove access to the work immediately and investigate your claim.

FACULTES UNIVERSITAIRES NOTRE-DAME DE LA PAIX
NAMUR
INSTITUT D'INFORMATIQUE

IMPLEMENTATION D'UN SYSTEME
DE GESTION DE FICHIERS
A STRUCTURE ARBORESCENTE

Mémoire présenté par
Patrick Majerus
en vue de l'obtention
du titre de
Licencié et Maître en Informatique.

ANNEE ACADEMIQUE 1982-1983

Je tiens à remercier Monsieur
Claude Cherton pour l'aide qu'il a ap-
portée à la réalisation de ce mémoire.

Je remercie également mon
épouse pour sa patience et le soutien
qu'elle m'a apporté.

TABLE DES MATIERES

Introduction.	1
1. Le projet.	2
2. But du mémoire.	5
Première partie : L'accès aux données.	6
1. Rappels. Position du problème.	7
2. De quels moyens disposons-nous?	9
2.1. La gestion par le système.	9
2.2. La gestion par le programmeur.	9
3. Comment utiliser ces moyens?	10
3.1. La gestion par le système.	10
3.2. La gestion par le programmeur.	10
3.2.1. Problèmes posés.	10
3.2.2. Solutions proposées.	11
3.2.2.1. Liste des blocs libres.	11
i) Première solution.	11
ii) Deuxième solution.	11
3.2.2.2. Position d'un enregistrement.	12
4. Conclusions. Choix d'une solution.	15
5. Développement de la solution choisie.	16
5.1. Que devons-nous faire?	16
5.2. Rappels.	16
5.3. Optimisations.	17
5.3.1. Table des blocs.	17
5.3.2. Buffer.	17
5.3.3. Accès aux blocs.	18
5.4. Structure de données.	18
5.5. Algorithmes.	19
5.5.1. Procédure READFILE.	19
5.5.2. Procédure WRITEFILE.	20

6. Structure de l'enregistrement.	22
6.1. Un enregistrement défini comme record variable.	22
6.2. Emploi d'une fonction de correspondance.	23
6.3. Choix d'une solution.	23
6.4. Développement de la solution.	24
6.4.1. Transfert de la table des noeuds.	24
6.4.2. Transfert de la table des places libres.	26
Deuxième partie : La lecture des commandes.	28
Troisième partie : Implémentation du système.	32
Introduction.	33
1. Le programme.	33
2. La disquette.	35
Quatrième partie : Les tests.	37
Introduction.	38
1. Procédures utilitaires de base.	39
2. Procédures de chargement et de déchargement des tables.	40
2.1. Table des places libres.	40
2.2. Table des noeuds.	41
2.2.1. Chargement.	41
2.2.2. Constitution de la table partielle.	46
3. Procédures utilitaires pour le traitement des commandes.	50
3.1. La procédure TRTARG.	50
3.2. La procédure TRTARG TRTMD.	53

4. Procédures de traitement des commandes.	63
4.1. La procédure TRTCRE.	64
4.2. La procédure TRTGEN.	67
4.3. La procédure TRTADD.	71
4.4. La procédure TRTINS.	76
4.5. La procédure TRTMOD.	78
4.6. La procédure TRTDEL.	79
 Annexe 1. Le texte du programme.	 81
 Annexe 2. Le manuel utilisateur.	 131
 Conclusion	 138

INTRODUCTION

1. Le projet.

Comme le renseigne son titre, ce mémoire a pour but d'implémenter un système de gestion de fichiers à structure arborescente. Ce système a fait l'objet d'une analyse détaillée dans le mémoire de Massimo BENEDETTI, "Etude d'un système de gestion de fichiers à structure arborescente." 1982.

Rappelons-en les principes et objectifs essentiels.

Les objectifs.

La documentation est une partie essentielle mais souvent négligée d'une application informatique. Essentielle parce que l'utilisateur, l'analyste chargé de la maintenance et le programmeur sont rarement une seule et même personne et qu'il est difficile, de comprendre un programme mal documenté à fortiori si on en n'est pas l'auteur.

Négligée à cause du travail long et fastidieux que cela représente pour le programmeur, qui ne dispose d'aucun outil pour l'aider.

La redondance de commentaires, par exemple, souvent nécessaire pour une bonne documentation n'est facilitée que par certaines fonctions des éditeurs.

Notre objectif est de réaliser un outil qui permette de gérer un programme comme un texte découpé en morceaux. Chaque morceau aura une identité propre et pourra faire l'objet de traitements particuliers, comme création, modification, recopiage en d'autres endroits à Les principes.

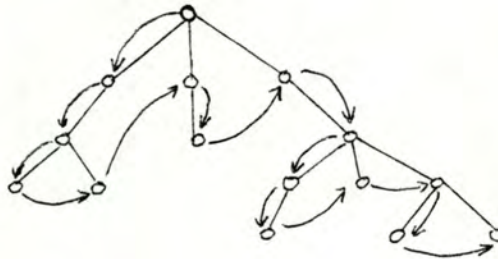
Ce système, destiné au traitement de textes, est basé sur une architecture arborescente : à chaque texte, ou morceau de texte, est associé un noeud d'un arbre. Les relations habituelles définies entre les noeuds d'une structure arborescente, racine, père, fils ou frère, se-

ront également applicables aux textes correspondants.

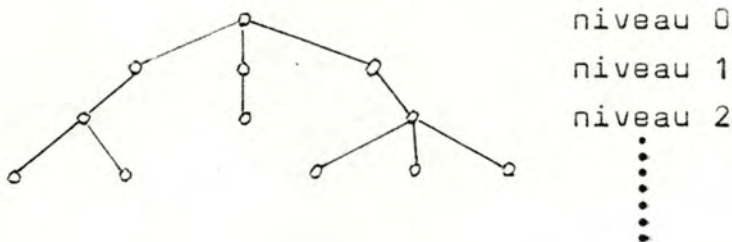
Notre application consistera donc en une gestion des noeuds, étant entendu qu'un noeud peut exister dans un arbre sans être associé à un texte.

Par la suite, nous nommerons "contenu du noeud" le texte associé à un noeud.

Un premier objectif a été de décrire une structure arborescente, ce qui signifie, pour chaque noeud, la possibilité de l'identifier et de retrouver son père, ses frères, ses descendants et la racine de l'arbre qui le contient. Pour cela, les noeuds seront ordonnés suivant le parcours dynastique. Ce que nous appelons parcours dynastique est indiqué par les flèches sur le schéma suivant :



L'arbre lui-même sera décomposé en niveaux, chaque noeud se trouvera sur un niveau particulier



Un noeud sera identifié par son nom et par le chemin à parcourir depuis la racine ou l'un de ses ascendants. Les fils et les frères d'un noeud seront obtenus en suivant le parcours dynastique. Deux pointeurs permettront d'atteindre directement le père d'un noeud et la racine de l'arbre qui le contient. Chacune des informations relatives à un noeud (nom, niveau, père, racine, suivant dans le parcours dynastique) entrera dans la constitution d'un descripteur de noeud. Les descripteurs des noeuds d'un arbre seront repris dans une table qui sera mémorisée sur fichier.

Un deuxième objectif a été de décrire le lien entre un noeud et le texte qui lui est éventuellement associé (son contenu). Ce contenu sera rangé dans un fichier le même que celui qui contient la table des noeuds. Le descripteur de noeud en question contiendra l'adresse dans le fichier du début du contenu, si celui-ci existe.

Un troisième objectif a été de pouvoir effectuer des opérations sur les noeuds et leur contenu.

On a voulu créer et insérer des noeuds,

supprimer des noeuds,

modifier des contenus de noeuds,

ajouter un contenu à celui déjà existant d'un
noeud,

générer, dans le contenu d'un noeud, le contenu d'un autre noeud.

Ces objectifs ont été atteints dans l'analyse organique du système.

2. But de ce mémoire.

Nous voulons disposer d'un système qui gère les données qui ont été définies, grâce aux traitements décrits par les procédures et à partir de commandes fournies par l'utilisateur. Au vu de ce qui a été fait, l'étape suivante, qui sera effectuée dans ce mémoire, consistera à

- réaliser l'accès aux données des fichiers
- réaliser la lecture des commandes de l'utilisateur
- tester et corriger les procédures de traitement afin d'en obtenir une version correcte.

Ces démarches constitueront les trois parties principales de cet ouvrage.

Dans la première, nous analyserons le problème de l'accès aux données sur les fichiers de l'application. A partir des caractéristiques du problème et des possibilités dont nous disposons sur Apple, nous essaierons de trouver une solution simple et adéquate.

Dans la deuxième partie, nous élaborerons une solution pour la lecture des commandes.

Une troisième partie, plus technique, décrira comment le système a été implémenté.

La quatrième partie tentera d'expliquer comment ont été effectués les tests, quels en ont été les résultats et quelles corrections il a fallu apporter aux textes des différentes procédures. Notre intention n'est pas de décortiquer toutes les erreurs rencontrées ; nous avons essayé de nous limiter aux erreurs les plus significatives, celles qui relevaient d'une lacune dans le traitement.

P R E M I E R E P A R T I E

L ' A C C E S A U X D O N N E E S

1. Rappels. Position du problème.

Chacun des fichiers de l'application correspond à une structure arborescente. Il contient les informations relatives à cette structure, c'est-à-dire la table des descripteurs des noeuds de l'arborescence et les contenus, s'ils existent, de chacun de ces noeuds. Nous voulons également qu'il contienne les informations concernant l'occupation de ses propres enregistrements (le mapping du fichier).

La table des noeuds, ainsi que les contenus des noeuds, sont de longueur quelconque, variable et tout à fait imprévisible. Il n'est donc pas possible de définir l'enregistrement du fichier de telle manière qu'il puisse contenir entièrement n'importe lequel des types d'information qu'il doit contenir.

Si nous choisissons une longueur fixe pour les enregistrements du fichier, il peut arriver que l'une ou l'autre des données que nous avons à ranger dans ce fichier va déborder sur plusieurs enregistrements. Ceci va nous poser un problème: comment retrouver les enregistrements qui recouvrent une même donnée?

Ce problème a déjà été résolu dans l'analyse fonctionnelle (cfr Massimo BENEDETTI "Etude d'un système de gestion de fichiers à structure arborescente" 1982).

Les enregistrements du fichier seront numérotés et contiendront, en plus des informations déjà citées, un pointeur.

Ce pointeur servira à chaîner entre eux les enregistrements du fichier qui couvrent une même donnée. Un enregistrement contiendra donc soit une partie de la table des noeuds, soit une partie du contenu d'un noeud, soit une partie du mapping du fichier et pointera vers la suite de la donnée, si elle existe (sauf s'il s'agit du mapping, auquel cas les enregistrements se suivent).

Le premier enregistrement est alors donné, pour la table des noeuds et le mapping du fichier, de manière fixe, et pour le contenu d'un noeud par un champ du descripteur de ce noeud (RCP: record pointer).

Un fichier de l'application est donc défini comme un ensemble d'enregistrements, rangés par ordre de numéro.

Le mapping du fichier est une table dont chaque entrée est un booléen qui indique si l'enregistrement qui a pour numéro l'indice de ce booléen est libre ou occupé.

Pour accéder à une donnée du fichier, il faudra accéder aux différents enregistrements qui la contiennent, s'il y a lieu, et ce en ne connaissant que leur numéro.

Le problème qui se pose à nous est donc le suivant: comment réaliser une structure de fichier qui nous permette d'atteindre un enregistrement sur base de son numéro?

2. De quels moyens disposons-nous?

2.1. La gestion par le système.

Nous disposons du système de gestion de fichiers de Pascal UCSD, implémenté sur APPLE II.

Si nous travaillons avec cet outil, nous devons déclarer chaque fichier, l'ouvrir, créer ses enregistrements, le fermer. La place en mémoire secondaire est attribuée statiquement aux enregistrements, qui sont rangés de manière contiguë. Les différents fichiers seront placés à la suite l'un de l'autre, de sorte que l'on ne peut étendre l'espace d'un fichier s'il est suivi d'un autre.

Ce dernier point nous obligera à réserver une fois pour toutes de l'espace en mémoire secondaire pour chacun des fichiers de l'application.

Une fois la réservation de place pour un fichier effectuée, il est possible d'accéder à ses enregistrements à partir de leur numéro, en se servant de la procédure SEEK, disponible en Pascal UCSD, qui positionne la fenêtre du fichier sur l'enregistrement de numéro donné.

2.2. La gestion par le programmeur.

En Pascal UCSD, le programmeur a la possibilité de contrôler lui-même les transferts physiques, notamment entre la mémoire centrale et la mémoire secondaire (disquette). Les procédures UNITREAD et UNITWRITE transfèrent en effet un bloc de 512 bytes de ou vers un buffer (tableau) déclaré en mémoire centrale.

3. Comment utiliser ces moyens?

3.1. La gestion par le système.

Si nous utilisons la gestion de fichiers de Pascal, la solution à notre problème est évidente, grâce à la procédure SEEK.

Malheureusement, cette solution présente un défaut très important: puisque l'espace d'un fichier n'est pas extensible, il faut réserver une certaine place à chaque fichier, sans savoir si cette place ne sera pas trop importante, ou bien insuffisante. Or, dans notre application, le nombre d'enregistrements effectivement occupés d'un fichier peut être fortement variable. On risque donc une forte perte de place en mémoire secondaire.

3.2. La gestion par le programmeur.

Cette approche, basée sur les procédures UNIT-READ et UNITWRITE, nous oblige à gérer nous-même l'espace d'un fichier en mémoire secondaire. Mais cela nous permettra peut-être d'éviter la perte de place en mémoire.

3.2.1. Problèmes posés.

L'unité d'espace sur la disquette, imposée par les procédures UNITREAD et UNITWRITE, est le block de 512 bytes. Suivant la longueur des enregistrements des fichiers, un bloc pourra en contenir un certain nombre. A chaque fichier, on veut attribuer, suivant ses besoins des blocs dans lesquels seront rangés ses enregistrements.

Il faut pour cela résoudre les deux problèmes suivants:

- quels blocs peuvent être attribués à un fichier sans en voler à un autre, c'est-à-dire quels sont les blocs libres, et quels sont ceux déjà occupés par un fichier?
- comment retrouver un enregistrement d'un fichier, c'est-à-dire comment retrouver le bloc qui le contient et la place qu'il occupe dans le bloc?

3.2.2. Solutions proposées

3.2.2.1. Liste des blocs libres

nous proposons ici deux solutions:

- i) Nous pourrions tenir à jour une table de booléens qui, pour chaque bloc, indique s'il est libre ou occupé. Cette solution occupera 35 bytes en mémoire centrale et sur la disquette (280 blocs/8 booléens par byte). Elle exigera aussi un parcours de la table pour chaque recherche d'un bloc libre.
- ii) Nous pourrions chaîner entre eux les blocs libres. Chaque bloc libre contiendra un pointeur vers le bloc libre suivant. L'adresse du premier bloc de la chaîne peut être retenue dans un pointeur particulier. On ne garde ainsi en mémoire centrale que le pointeur de départ de la chaîne, qui fournit immédiatement l'adresse d'un bloc libre.

Conclusions.

Cette deuxième solution nous semble préférable puisqu'elle n'occupe que très peu de place en mémoire centrale (un entier) et en mémoire secondaire, puisque les blocs libres dans lesquels figurent les pointeurs sont de toute manière inutilisés.

La recherche d'un bloc libre ne demandera plus de parcours en table.

Par contre, la libération d'un bloc demandera un accès physique.

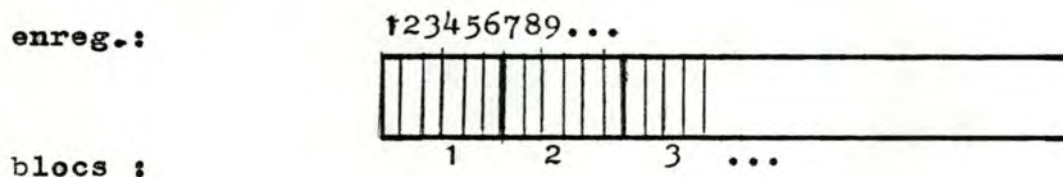
3.2.2.2. Position d'un enregistrement

Les enregistrements d'un fichier sont numérotés, et nous avons dit plus haut que nous devons pouvoir accéder à un enregistrement en fonction de son numéro. La position d'un enregistrement en mémoire secondaire devra donc dépendre de son numéro.

La longueur de l'enregistrement étant fixée, nous pouvons en déduire qu'un bloc de 512 bytes peut contenir N enregistrements. Un fichier ne pouvant occuper que l'espace de Nb blocs disponibles sur la disquette contiendra au maximum $N \cdot Nb$ enregistrements.

Nous convenons qu'un fichier contient exactement ce nombre d'enregistrements, mais qu'un enregistrement peut être libre ou occupé (ce qui est retenu par le mapping du fichier).

Découpons le fichier en blocs logiques contenant chacun N enregistrements et numérotons les dans l'ordre de la numérotation des enregistrements, suivant le schéma ci-dessous:



Puisque les enregistrements sont de longueur fixe, on peut calculer le numéro de bloc à partir du numéro d'enregistrement:

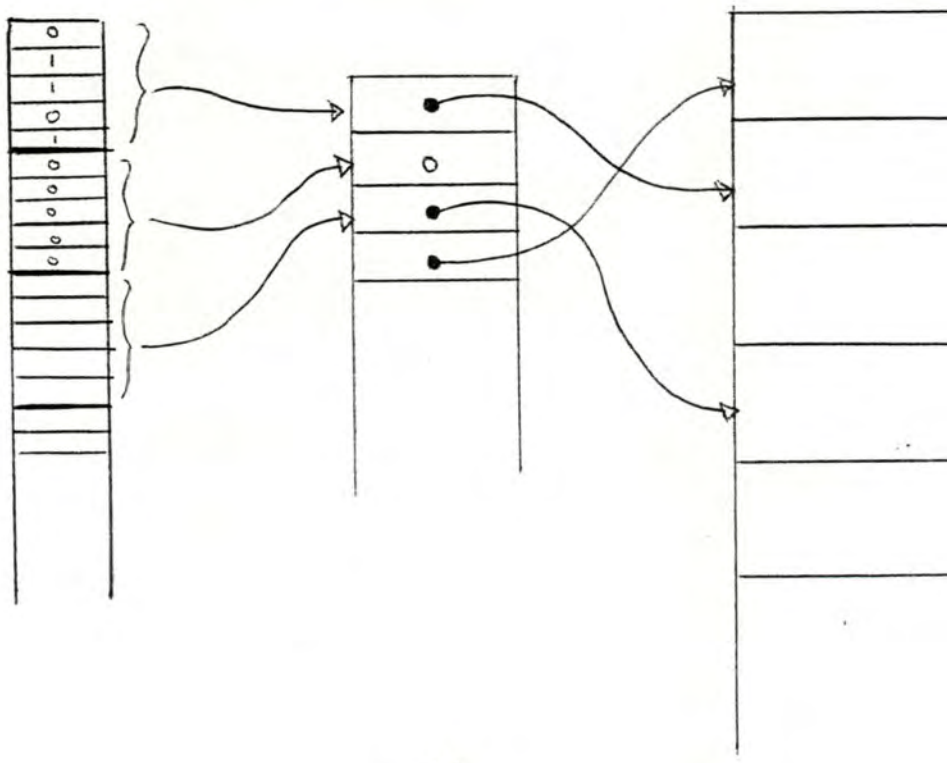
$$\text{numblc} = (\text{numenr} \text{ DIV } N) + 1$$

où DIV est la division entière.

Un bloc logique est occupé si un au moins de ses enregistrements est occupé. Dans ce cas, il doit correspondre à un bloc physique. Cette correspondance peut être établie, par exemple, à l'aide d'une table de Nb entrées où l'entrée correspondant au bloc logique contient l'adresse du bloc où sont effectivement rangés les enregistrements de ce bloc logique.

Si, par contre, aucun enregistrement du bloc logique n'est occupé, alors ce bloc logique ne correspond à aucun bloc physique, et l'entrée de la table peut être mise à zéro.

mapping du fichier table des blocs logiques disquette



A chaque fichier de l'application correspond une table des blocs qui indique quels sont les blocs occupés ainsi que leur adresse sur la disquette.

A partir du numéro d'un enregistrement, on peut donc facilement déterminer le bloc dans lequel il se trouve. Il faut encore pouvoir déterminer sa position dans le bloc.

Puisqu'un bloc contient N enregistrements de numéros successifs, il suffit de prendre comme numéro dans le bloc le reste de la division du numéro de l'enregistrement par N.

Cette solution offre l'avantage d'être simple à implémenter, puisque deux formules simples suffisent pour calculer la position de l'enregistrement, et d'être rapide au niveau des accès physiques, un seul accès étant nécessaire pour un bloc dont l'adresse est déterminée par calcul et par consultation d'une table en mémoire centrale.

Elle demande par contre de la place: une table de blocs par fichier en mémoire secondaire et la table du fichier en cours de traitement en mémoire centrale (une pareille table contient $2 \cdot N_b$ bytes).

D'autre part, il peut y avoir perte de place à l'intérieur d'un bloc, du fait que tous les enregistrements du bloc ne sont pas nécessairement occupés.

On pourrait essayer de limiter cette perte de place en mémoire en supprimant l'intermédiaire de la table des blocs logiques et en rangeant directement un enregistrement à sa place dans le bloc physique dont on aurait obtenu l'adresse par calcul. Mais si on travaille avec plusieurs fichiers, autant d'enregistrements devront occuper le même bloc et la même place dans ce

bloc, puisqu'ils sont numérotés de la même façon dans les différents fichiers. Il faudra ranger ces enregistrements ailleurs qu'à leur "bonne" place, en se donnant un moyen de les retrouver. Ceci risque de faire augmenter la place occupée et, surtout, d'augmenter le nombre des accès physiques nécessaires par accès à un enregistrement.

4. Conclusions. Choix d'une solution.

Nous avons envisagé trois solutions pour la gestion des fichiers:

- i) utiliser la gestion de fichiers de Pascal
- ii) laisser la gestion au programmeur, avec table de blocs intermédiaire
- iii) laisser la gestion au programmeur, sans table de blocs intermédiaire.

La première solution ne nous permet pas d'avoir des fichiers de taille variable et peut, de ce fait, entraîner une grande perte de place en mémoire secondaire.

La deuxième solution, même si elle demande l'intervention du programmeur pour la gestion des transferts est simple à implémenter et rapide. Elle occupe cependant de la place en mémoire, où il peut y avoir des pertes.

La troisième solution récupère la place perdue par la deuxième, mais, en contrepartie, le nombre d'accès physiques va augmenter, de même que la complexité de l'implémentation.

Etant donné ces circonstances, c'est la deuxième solution que nous allons choisir et développer, la place perdue étant trop peu importante pour justifier une augmentation du temps d'accès.

5. Développement de la solution choisie.

5.1. Que devons-nous faire?

Nous devons écrire deux procédures:

- i) une procédure de lecture d'un enregistrement dont on donne le numéro et appartenant à un fichier dont on donne le nom ou le numéro,
- ii) une procédure d'écriture d'un enregistrement dont on donne le numéro et appartenant à un fichier dont on donne le nom ou le numéro.

5.2. Rappels. (cfr 3.2.)

Nous disposons, pour chaque fichier, d'une table des blocs dont les entrées sont, s'ils existent, les adresses des blocs physiques occupés par le fichier.

D'autre part, les blocs libres de la disquette sont chaînés entre eux et l'adresse du premier est disponible dans une variable en mémoire centrale.

Remarque:

La disquette étant constituée de 280 blocs, chaque fichier peut potentiellement contenir 280 blocs. La table des blocs de chaque fichier doit donc être une table de 280 entiers, c'est-à-dire de 560 bytes.

Nous devons mémoriser ces tables en mémoire secondaire, sur des blocs de 512 bytes. Avec des tables de 560 bytes, nous allons soit perdre de la place,

soit compliquer la gestion.

C'est pour cette raison que nous choisissons de travailler avec des tables de 512 bytes, soit 256 entiers. La taille maximale d'un fichier est donc limitée à 256 blocs.

5.3. Optimisation

5.3.1. Table des blocs.

La table des blocs de chaque fichier doit être mémorisée en mémoire secondaire. D'autre part, elle doit se trouver en mémoire centrale chaque fois qu'on veut accéder à un enregistrement du fichier correspondant. On pourrait garder ces tables en permanence en mémoire centrale, mais alors on occupe une table pour chaque fichier de l'application.

Il est plus économique de n'avoir qu'une table en mémoire, celle du fichier auquel on doit accéder. On doit alors charger la table du fichier en mémoire si celui-ci n'est pas celui qui a été accédé en dernier lieu. On augmente ainsi le nombre des accès physiques par accès logique, mais pas dans des proportions importantes si on ne saute pas d'un fichier à l'autre.

5.3.2. Buffer.

Pour effectuer les transferts, nous devons nous donner un buffer en mémoire centrale, c'est-à-dire une table de ou vers laquelle seront transférés le ou les blocs de la disquette. Etant donné la manière dont nous gérons les fichiers, il nous semble préférable de prendre un buffer de la taille d'un bloc (les enregistrements sont rangés dans des blocs logiques de même taille que les blocs physiques).

5.3.3. Accès au bloc

Pour chaque lecture ou écriture, on peut transférer dans le buffer le bloc concerné. Mais ce bloc s'y trouve peut-être déjà. Dans ce cas, le transfert n'est pas nécessaire. Si par contre le transfert est nécessaire et si le contenu actuel du buffer a été modifié, il faudra recopier le bloc qui se trouve dans le buffer avant d'y amener le nouveau bloc.

Deux variables peuvent suffire pour réaliser cela: une variable entière qui retient l'adresse du bloc dans le buffer,
une variable booléenne qui retient si le contenu du buffer a été ou non modifié.

Si l'opération porte sur un autre fichier, il faut charger la table des blocs de ce fichier. Le bloc dans le buffer n'est pas le bon puisqu'il appartient à un autre fichier. Il faudra donc charger le nouveau bloc. Il suffit d'une variable entière pour retenir le numéro du fichier actuellement en cours de traitement et dont la table des blocs est en mémoire.

5.4. Structure des données.

Nous pouvons donner la structure de donnée sur laquelle vont travailler les deux procédures de lecture et d'écriture, en plus de la structure de donnée de l'application.

Var flenum: integer; numéro du fichier à consulter
 prcfile: integer; numéro du fichier précédemment consulté
 enrnum: integer; numéro de l'enregistrement à lire ou à écrire
 blknum: integer; numéro du bloc logique contenant cet enregistrement
 blkadr: integer; adresse du bloc physique correspondant au bloc logique
 prcblk: integer; adresse du bloc physique se trouvant dans le buffer
 prcop : boolean; est vrai si le contenu du buffer a été modifié, faux sinon
 flebuf: array [1..N] of enregistrement;
 buffer pour le transfert des blocs
 tampon: enregistrement;
 variable de ou vers laquelle on veut transférer l'enregistrement
 ptrlst: array [1..256] of integer;
 table des blocs du fichier consulté

5.5. Algorithmes.

5.5.1. Procedure READFILE.(flenum,enrnum,tampon)

READFILE lit l'enregistrement de numéro enrnum du fichier de numéro flenum et place son contenu dans tampon.

if flenum \neq prcfile

then begin

 charger la table des blocs de flenum;

 prcfile:=flenum

end;

blknum:=enrnum DIV N + 1; calcul du numéro de bloc à partir du numéro d'enregistrement


```

blkadr:=ptrlst[blknum]; consultation de la table des
                        blocs pour déterminer l'adres-
                        se du bloc physique contenant
                        l'enregistrement

if blkadr ≠ prcblk
then
                        le bloc ne se trouve pas dans
                        le buffer

begin
  if prcop
  then
                        l'ancien bloc a été modifié
begin
  prcop:=false; le nouveau bloc n'a pas enco-
re été modifié
  UNITWRITE(flebuf,prcblk);
                        on écrit le contenu de flebuf
                        dans le bloc prcblk
end;
  preblk:=blkadr; le buffer contient un nouveau
                        bloc
  UNITREAD(flebuf, blkadr)
                        le contenu du bloc blkadr est
                        transféré dans le buffer
end;
enrnum:=enrnum MOD N + 1; calcul de la position de l'en-
                        registrement dans le bloc
tampon:=flebuf[enrnum]

```

5.5.2. Procedure WRITEFILE (flenum, enrnum, tampon)

```

WRITEFILE écrit l'enregistrement de numéro
enrnum dans le fichier de numéro flenum, à partir de
la variable tampon.

if flenum ≠ prcfile
then begin
  charger la table des blocs du fichier flenum;
  prcfile:=flenum
end;

```

```

blknum:=enrnum DIV N + 1;   calcul du numéro de bloc à
                             partir du numéro d'enregist-
                             rement
blkadr:=ptrlst[blknum];     consultation de la table des
                             blocs
if blkadr = 0
  then chercher un bloc libre;
if blkadr = 280
  then ERREUR               il n'y a plus de bloc libre
  else
    begin
      ptrlst[blknum]:=blkadr;  il faut mettre à jour la
                               table des blocs qui indi-
                               que les blocs occupés
      if blkadr≠preblk
        then
          begin
            if prcop
              then UNITWRITE(flebuf, preblk);
            preblk:=blkadr;
            UNITREAD(flebuf, blkadr)
          end;
                               on met le bloc à modifier
                               dans le buffer s'il ne s'y
                               trouve pas
      enrnum:=enrnum DIV N + 1;
      flebuf[enrnum]:=tampon;
      prcop:=true;           on écrit le contenu de
                               tampon dans le buffer,
                               et on indique que le bloc
                               a été modifié
    end

```


6. Structure de l'enregistrement du fichier.

Nous avons déjà vu plus haut qu'un enregistrement était constitué d'une partie "donnée" et d'un pointeur servant à chaîner les enregistrements couvrant une même donnée. (cfr 1. Rappels)

Nous avons vu aussi qu'un enregistrement devait pouvoir contenir trois types de donnée différents:
une partie de la table des noeuds, c'est-à-dire un certain nombre de descripteurs de noeud,
une partie du mapping du fichier, c'est-à-dire une table de booléens,
une partie du contenu d'un noeud, c'est-à-dire une table de caractères.

On peut envisager deux possibilités pour réaliser cette structure:
soit définir l'enregistrement comme un record variable,
soit choisir l'enregistrement d'un type particulier et se servir d'une fonction de correspondance entre le type de l'enregistrement et celui des différentes données.

6.1. Un enregistrement défini comme record variable.

Le champ destiné aux descripteurs de noeud sera une table de records, chaque record étant un descripteur; L'enregistrement pourrait donc être du type

```
ENR = record of inrec:REC;  
      nxr:integer  
end;
```

```

où REC = record case typ:integer of
    1:(array [1..nse] of DESCR);
    2:(array [1..idtmxl] of char);
    3:(array [1..blnmxl] of boolean)
end
où DESCR = record
    nnm: packed array of char;
    fap: integer;
    lnb: integer;
    nxp: integer;
    rcp: integer;
    rtp: integer
end

```

6.2. Emploi d'une fonction de correspondance.

Pascal UCSD met à notre disposition des procédures de transfert de bytes, MOVELEFT et MOVERIGHT. Ces procédures permettent de transférer une variable dans une autre, indépendamment du type de ces deux variables.

MOVELEFT(a, b, c) a pour effet de transférer c bytes de a vers b, en commençant par la gauche.

6.3. Choix d'une solution.

Remarquons que les transferts impliquant la table des noeuds et le mapping du fichier se font en début et en fin de traitement, alors que ceux qui impliquent des contenus de noeuds font partie intégrante des traitements.

On pourrait donc envisager de favoriser ce dernier type de transfert au point de vue de la vitesse et de la simplicité, par rapport aux deux autres types. Ainsi, si on déclare un enregistrement comme un tableau de caractères, le contenu d'un noeud pourra être transféré sans correspondance ni test, comme ce serait le cas avec une structure de record variable. La correspondance devrait alors être faite pour les autres données, pour lesquelles on aurait un traitement plus complexe.

Nous avons opté pour cette solution car elle nous paraît également plus claire.

6.4. Développement de la solution.

Il y a deux cas à envisager: le transfert de la table des noeuds et le transfert du mapping du fichier.

6.4.1. Transfert de la table des noeuds

Les procédures affectées par la correspondance sont les procédures SETNTBENT et SETREC, qui effectuent le transfert entre l'enregistrement et la table, dans les deux sens. Nous nous contenterons de développer SETNTBENT, l'autre procédure étant tout à fait analogue.

Rappelons la structure d'un descripteur de noeud:

```
NTBENT = record
    NNM: packed array [1..NNMMXL] of char;
    LNB: integer;
    RTP: integer;
    NXP: integer;
    FAP: integer;
    RCP: integer
end
```

Dans SETNTBENT, la variable entière I désigne la position, dans l'enregistrement, à partir de laquelle est écrit le descripteur. La variable entière J désigne la dernière position occupée par le champ NNM du descripteur dans l'enregistrement. Pour ce champ, il n'y a pas de correspondance à faire, le recopiage est immédiat:

```
nnmcrc:=1;
for idtcrc:=I to J do
  begin
    ntb[ntbcrc].nnm[nnmcrc]:=stk[stkcrx].buf.idt[idtcrc];
    nnmcrc:=nnmcrc+1
  end;
```

Rappelons que l'enregistrement a d'abord été transféré en stk.buf.idt, avant que son contenu ne soit transformé pour constituer les descripteurs de noeud.

Nous devons traduire les caractères qui représentent les pointeurs d'un descripteur en type entier. Un entier, sur Apple II, occupe deux bytes et peut donc être transféré par MOVELEFT dans une variable de deux caractères, par exemple un tableau de deux caractères. Les deux caractères qui représentent un entier seront d'abord transférés dans ce tableau. Par MOVELEFT, on transférera alors ce tableau dans la variable entière voulue.

Nous nous donnons donc une variable déclarée comme suit:

```
var conversion: packed array [1..2] of char;
```

Le transfert vers la table des noeuds peut se faire, pour l'entier LNB par exemple, de la manière suivante:


```

I:=J+1;
J:=J+2;                on positionne I et J sur les deux ca-
                        ractères de stk.buf.idt qui suivent
                        le champ NNM et qui sont représenta-
                        tifs de LNB.

conversion[1]:=stk[stkcrx].buf.idt[I];
conversion[2]:=stk[stkcrx].buf.idt[I];
MOVELEFT( conversion, ntb[ntbcrx].lnb, 2);

```

Les instructions à exécuter pour transférer les autres champs d'un descripteur sont identiques à celles-ci.

6.4.2. Transfert de la table des places libres.

Les bnb premiers enregistrements du fichier sont réservés à cette table.

Le nombre de booléens qu'on peut ranger dans ces enregistrements est

$$\text{bnb} * \text{blnmxl}$$

où blnmxl est la longueur en bits d'un enregistrement.

Un fichier peut contenir $N * 256$ enregistrements, où N est le nombre d'enregistrements par bloc. Le mapping du fichier est donc une table de $N * 256$ booléens.

Il se peut que $\text{bnb} * \text{blnmxl}$ soit supérieur à $N * 256$. Dans ce cas, un traitement spécial est à prévoir pour le dernier des bnb enregistrements, qui contient alors trop de booléens.

Ici aussi, nous nous servons d'une variable intermédiaire, car nous ne pouvons pas utiliser comme paramètre de MOVELEFT une partie d'un tableau. Cette variable peut être déclarée de la manière suivante:

```

var conversion: packed array 1..blnmxl of boolean;

```

On transférera dans cette variable, en faisant la correspondance, tout le contenu de l'enregistrement, avant de le reporter à la bonne place dans la table.

```

I:=0;                                I sert à compter les enregist-
                                     trements lus

repeat
  READFILE( flenum, I, stk.buf.idt);
                                     lecture de l'enregistrement de
                                     numéro I du fichier
  MOVELEFT(stk.buf.idt, conversion, idtmxl);
                                     conversion des caractères en
                                     booléens

  for j:=1 to blnmxl do
    ftb[ftbcrx][I*blnmxl+j-1] :=conversion j ;
                                     I, le numéro de l'enregistre-
                                     ment, sert à déterminer l'in-
                                     dice de la table à partir du-
                                     quel il faut recopier les boo-
                                     léens représentés dans cet
                                     enregistrement

    I:=I+1;

until I = bnb-1;                    les bnb premiers enregistre-
                                     ments sont recopiés intégrale-
                                     ment

k:=((fplmxl+1) MOD blnmxl);
                                     k contient le nombre effectif
                                     de booléens dans le dernier
                                     enregistrement

READFILE( flenum, I, stk.buf);
MOVELEFT( stk.buf.idt, conversion, idtmxl);
for j:=1 to k do
  ftb[ftbcrx][((bnb-1)*blnmxl+j-1] :=conversion j ;

```

Ces instructions effectuent le transfert vers la mémoire centrale. Le transfert inverse se fait de manière identique.

DEUXIEME PARTIE

LA LECTURE DES COMMANDES

Nous disposons d'un fichier, appelé fichier de commande, dans lequel nous devons aller lire, séquentiellement, des commandes écrites par l'utilisateur.

Notre problème est de définir la structure de ce fichier ainsi que les procédures de lecture et d'écriture des commandes. Une commande est une suite de caractères, de longueur quelconque. Ceci ne nous permet pas de définir un fichier Pascal tel que tout enregistrement contienne exactement une commande, à moins d'imposer une longueur maximale à une commande.

Si cette solution était adoptée, la création d'un noeud avec un contenu relativement long pourrait exiger plusieurs commandes: une commande CREATE suivie du nombre de commandes ADD nécessaires.

Nous avons décidé d'adopter une solution qui autorise une longueur quelconque qui n'impose donc pas de limite. Il n'est alors pas possible de définir une commande comme un enregistrement du fichier de commande.

On pourrait découper une commande en ligne; chaque ligne, de longueur fixe, étant alors un enregistrement du fichier

Il faut donner à l'utilisateur un moyen de créer les enregistrements. Un programme qui lit une suite de caractères à l'écran et qui les recopie successivement dans le buffer du fichier serait une solution. Mais cela obligerait l'utilisateur à remplir ce buffer, même si la ligne de commande est plus courte, ce qui est assez fastidieux. Il faudrait pouvoir détecter la fin de la ligne de commande et remplir automatiquement le buffer avec des blancs si la ligne de commande est trop courte. On peut décider, par exemple, que le caractère "carriage return" indique une fin de ligne et provoque le remplissage du buffer. De cette manière, l'utilisateur

peut écrire des lignes de commande de longueur quelconque, inférieure à celle du buffer.

Etant donné qu'une commande peut être formée de plusieurs lignes, il faut pouvoir en détecter la fin. Si elle comporte un contenu, sa fin correspond à celle de ce contenu. Sinon, on peut l'indiquer par un caractère inutilisé dans la syntaxe des commandes. Nous avons choisi ";".

Plutôt que de créer un programme à part qui effectue la lecture des lignes de commande, nous avons choisi d'implémenter ce traitement à l'intérieur de l'application, chaque ligne étant constituée à partir d'une suite de caractères au moment de son traitement. Le fichier de commande devient alors un fichier de caractères, copie de ce que l'utilisateur a tapé à l'écran, et obtenu par l'éditeur disponible sur Apple.

Nous pouvons écrire la procédure READFØ, de lecture d'une ligne de commande. Cette procédure a pour seul paramètre la variable "tampon" de type packed array [1..idtmxl] of char, dans laquelle doit être copiée la ligne de commande.

```
Procedure READFØ(tampon);
```

```
begin
```

```
  for idtcx:=1 to idtmxl do
```

```
    tampon[idtcx] := ' ';
```

```
  idtcx:=0;
```

repeat

idtcx:=idtcx+1;

tampon[idtcx]:=FØ↑;

if(idtcx≠1) OR NOT EOLN(FØ)

then get(FØ)

si on ne fait pas ce test et
si deux "carriage return" se
suivent, on décale d'un carac-
tère les lignes suivantes

until (idtcx=idtmx1) OR EOLN(FØ);

on remplit le tampon jusqu'à
ce qu'il soit plein ou jus-
qu'à la fin de la ligne de
commande

end

Remarque.

Pour initialiser la procédure, tout appel doit être pré-
cédé d'un GET(FØ).

TROISIEME PARTIE

IMPLEMENTATION DU SYSTEME

Introduction.

Dans le premier chapitre, notre but est d'établir la structure du programme de telle sorte qu'elle facilite et accélère la compilation.

Dans le second chapitre, nous décrirons la structure de la disquette qui doit servir de support aux fichiers de l'application.

1. Le programme.

Notre objectif est de pouvoir compiler l'ensemble des procédures de telle manière qu'une modification dans une procédure ne nous oblige pas à les recompilier toutes. Pour réaliser cet objectif, Pascal UCSD permet le travail avec des librairies, unités de textes compilables séparément et pouvant être introduites dans les programmes objets d'autres procédures.

Un programme peut utiliser plusieurs librairies, une librairie peut elle-même utiliser d'autres librairies.

Le problème est maintenant de savoir comment constituer les librairies.

Dans notre application, il y a trois types de procédures:

les procédures utilitaires d'accès aux données des fichiers,

les procédures d'initialisation et de terminaison,

les procédures de traitements des commandes.

Dans une première librairie, nous allons déclarer toutes les procédures élémentaires, plus TRTNID, qui est une procédure commune à plusieurs traitements. Elle contiendra également les déclarations des constantes, des types et des variables globales.

Nous appelons cette librairie " DECL ".

Dans une seconde librairie, nous déclarons les procédures d'initialisation et de terminaison. Cette librairie, INTER, utilise la librairie DECL, puisque ses procédures utilisent les déclarations de variables et de procédures de DECL.

Une troisième librairie, CONT, contiendra certaines procédures de traitement communes à différentes commandes, par exemple TRTCONNOD et TRTCONNODVAL.

Enfin, les procédures de traitement de commandes, TRTCRE, TRTADD, ..., seront déclarées dans le programme principal.

A partir des versions codées du programme principal et des différentes librairies, le linker d'Apple II va créer un programme exécutable. Ce programme sera stocké sur une diskette, dans un drive, le second drive étant laissé libre pour le support des fichiers.

2. La disquette.

Nous savons déjà que :

les blocs non occupés par un fichier sont chaînés entre eux au moyen de pointeurs, cette disquette doit en permanence contenir les tables des blocs de tous les fichiers de l'application.

D'autre part, étant donné que le système accède à cette disquette avec les procédures UNITREAD et UNITWRITE, il faudra un moyen d'identifier cette disquette et de vérifier sa présence dans le drive, cela, surtout pour éviter de détruire des blocs d'information sur d'autres disquettes.

Nous allons réserver le premier bloc (adresse 0) à cette identification, qui se restreint pour le moment à un nom.

Les quatre blocs suivants (adresses de 1 à 4) seront destinés à recevoir les tables des blocs des quatre fichiers de l'application.

Le premier bloc libre est donc celui d'adresse 5. Cette adresse, qui peut être modifiée en cours de traitement, est une propriété de la disquette et doit donc être mémorisée sur la disquette. Nous pouvons la mémoriser dans le bloc 0 où il nous reste de la place.

A partir de ce premier bloc, il faudra chaîner tous les autres blocs libres (initialement, tous les blocs de la disquette depuis 5 jusqu'à 279).

Nous pouvons écrire une procédure d'initialisation de la disquette.

Les spécifications seront:

- la lecture à l'écran du nom de la disquette et la mémorisation dans le bloc 0.
- l'initialisation, dans le bloc 0, du pointeur vers le premier bloc libre, à la valeur 5.
- la mise à 0 de toutes les entrées des tables des blocs dans les blocs 1,2,3 et 4.
- la mise à I+1 du pointeur du bloc I pour I compris entre 5 et 279.

Pour libérer un bloc d'adresse I, il faudra:

- donner au pointeur du bloc I l'ancienne valeur du pointeur vers le premier bloc libre.
- mettre à la valeur I le pointeur vers le premier bloc libre.

L'occupation d'un bloc libre I va entraîner le forçage du pointeur vers le premier bloc libre à la valeur du pointeur de I, vers le suivant de I dans la chaîne des blocs libres.

QUATRIEME PARTIE

LES TESTS

Introduction.

Nous avons devant nous un gros système qu'il faut tester. Il est évidemment impossible de tout tester d'un coup, il faut faire une découpe de ce système et tester chaque morceau séparément.

Nous avons décidé de ne pas tester certaines procédures, vu leur simplicité.

D'autres procédures sont constituées (comme TRTNID, par exemple) de procédures assez simples et d'une ou deux autres plus complexes. On peut considérer que celles-ci sont des procédures élémentaires utilisées par une autre procédure pour effectuer le traitement d'une commande, ou d'une partie d'une commande.

Nous nous contenterons de tester ces procédures de traitement dans leur ensemble.

Le plan des tests s'établit donc comme suit :

- Procédures utilitaires de base.

MOVELEFT, UNITWRITE, UNITREAD

- Procédures de chargement et de déchargement des tables

- Procédures utilitaires pour le traitement des commandes

TRTARG, TRTNID

- Procédures de traitement des commandes

TRTCRE, TRTGEN, TRTINS, TRTMOD, TRTADD, TRTDEL

1. Procédures utilitaires de base.

Ces procédures sont des procédures de transfert. Il est impossible de tester UNITWRITE indépendamment de UNITREAD puisqu'on n'a pas accès au contenu de la disquette autrement que par UNITREAD.

Nous avons donc testé ces deux procédures en faisant faire l'aller retour entre la mémoire centrale et la disquette à une chaîne de caractères. En comparant la chaîne de départ à la chaîne d'arrivée les résultats des tests ont été satisfaisants et nous ont permis de faire confiance à ces procédures.

Pour tester la procédure MOVELEFT, nous devons tester si les transformations booléen-caractère et entier-caractère se font correctement dans les deux sens.

On peut tester le transfert d'une variable entière vers une variable packed array of char, in vérifiant que les caractères obtenus sont bien ceux dont le code est identique aux deux bytes de la représentation de l'entier. Mais qu'en est-il si le contenu de ces deux bytes ne correspond pas à un code caractère ?

Nous avons alors fait un test aller-retour, en transférant l'entier dans le tableau de deux caractères puis en retransférant ce tableau dans une autre variable entière. La comparaison des valeurs entières de départ et d'arrivée permet de conclure que la procédure MOVELEFT fonctionne correctement.

2. Procédures de chargement et de déchargement des tables.

La procédure de chargement des tables reconstitue, en mémoire centrale, la table des noeuds et la table des places libres de chaque fichier.

Pour chaque fichier non vide de l'application, la procédure va lire les enregistrements de la table des noeuds qui se trouvent dans ce fichier et dont on connaît le numéro du premier (une constante) . Les descripteurs de noeud contenus dans ces enregistrements sont placés dans la table des noeuds au fur et à mesure qu'on les rencontre.

De même, pour chaque fichier non vide, la procédure lit les enregistrements qui contiennent la table des places libres du fichier et constitue une table des places libres en mémoire centrale.

La procédure de rechargement recopie les tables en mémoire secondaire. Chacune des tables des places libres est recopiée dans les enregistrements du fichier correspondant.

Parmi les descripteurs de la table des noeuds, la procédure extrait ceux des noeuds d'un même fichier et constitue une table triée suivant l'ordre dynastique. Cette table est alors transférée dans les enregistrements du fichier correspondant.

2.1. Procédures de chargement et rechargement de la table des places libres.

Nous ne testerons pas ces procédures étant donné qu'elles sont essentiellement constituées d'un transfert

entre mémoires centrale et secondaires, et d'une transformation par la procédure MOVELEFT.

Nous devons cependant signaler une chose importante. La table des places libres d'un fichier n'est chargée en mémoire centrale que si ce fichier n'est pas vide. Si le fichier est vide, rien n'est prévu. Il suffit, pour remédier à cela, d'introduire une petite procédure qui met à "true" toutes les entrées de la table des places libres de ce fichier.

2.2. Procédures de chargement et rechargement de la table des noeuds.

Nous avons en fait deux choses à tester :

- Une table des noeuds qui est rangée sur la disquette puis chargée en mémoire centrale est identique à la table donnée au départ.
- La sélection des noeuds d'un fichier et leur mise en ordre dynastique.

2.2.1. Chargement de la table.

Nous aurons besoin, pour tester le bon fonctionnement du chargement et du rechargement, de tester ces deux procédures ensemble.

En effet, nous devons disposer, dans le fichier, d'une table de noeuds, que nous aurons dû y enregistrer au préalable. Comme nous disposons d'une procédure qui effectue cet enregistrement, nous pouvons l'utiliser. Si, par contre, il y a des différends, c'est qu'une des procédures (ou les deux) comporte des erreurs. Il sera évidemment plus difficile de détecter ces erreurs. Considérons le texte des procédures à tester pour éventuellement

y déceler des erreurs et pour établir les tests à effectuer.

Procedure RCHNTB (var crtadr : integer)

```
begin
  pntcrx := 1;
  crtngx := bnb;
  repeat
    SETREC;
    if pntcrx = 0 then
      begin
        rec.nxr := 0;
        WRTNTB (crtngx)
      end
    else
      begin
        FNDFPL (rec.nxr);
        WRINTB (crtngx)
      end
    end
  until pntcrx = 0
end
```

Cette procédure transforme les descripteurs de noeuds pour les ranger dans un enregistrement du fichier (par SETREC, qui est une procédure analogue à SETNTBENT, que nous avons decrite plus haut). Ensuite, elle cherche éventuellement une place libre dans le fichier pour y ranger les enregistrements ainsi constitués. Elle ne présente pas de grosse difficulté si ce n'est la différence

de traitement entre le dernier enregistrement et le premier, pour lequel on ne cherche pas de place libre.

On pourrait effectuer les tests suivants :

- Une table d'au plus NSE noeuds (un seul enregistrement) pour passer directement dans le premier cas. On ne passe alors jamais dans le deuxième chemin.
- Une table de plus de NSE noeuds (au moins deux enregistrements).

On passe alors au moins une fois par le deuxième chemin.

Procedure CHGNTB (var crtfnm : packed array
 [1..nnmmx1] of char)

```
Var i,j,k : integer;
    fstidx,lstidx : 0..ntbmx1;
begin
    k := 1 ; i := 1 ; j := nnmmx1;
    stk[stkcrx].fnm := crtfnm;
    stk[stkcrx].nxr := bnb;
    RDLIN(stk[stkcrx]);
    while k < nse + 1 do
        begin
            ntbcx := ntbcx + 1;
            SETNTBENT;
            i := j + 1;
            j := j + nnmmx1;
            if k = 1
            then
                begin
                    rtb[rtbcx].ntbptr := ntbcx;
                    ntb[ntbcx].fap := rtbcx;
                    fstidx := ntbcx
                end;
        end;
```



```

        k:=k+1
    end
repeat
    RDLIN(stk[stkcrx]);
    k:=1; i:=1; j:=nnmmx1;
    while k ≠ nse +1 do
        begin
            ntbcx:=ntbcx+1;
            SETNTBENT;
            i:=j+1; j:=j+nnmmx1;
            if stk stkcrx .nxr ≠ 0
                then k:=k+1
                else k:=nse+1
            end
        until stk[stkcrx].nxr = 0; ...

```

Remarquons d'abord que la table des noeuds est entièrement chargée en mémoire centrale et que c'est sur cette table en mémoire que s'effectueront toutes les opérations. En fin de travail, la table éventuellement modifiée sera rechargée en mémoire secondaire. Au moment du chargement, il faudra donc libérer tous les enregistrements qui étaient occupés par la table, pour éviter qu'il n'y ait, à la fin du travail, deux versions de la table des noeuds dans le fichier. Chaque enregistrement lu doit donc être libéré, c'est-à-dire que l'entrée correspondante de la table des places libres du fichier doit être mise à "true". Ce n'est cependant pas nécessaire pour le premier enregistrement de la table, puisqu'il occupe toujours la même place dans le fichier.

Nous ne connaissons pas ici l'indice de la table des places libres du fichier que nous traitons, nous ne connaissons que le nom du fichier dans la variable `fsk[fskcrx]`. La procédure FREE permet de retrouver cet indice et en même temps de mettre à jour la table:

```

Procedure FREE (var flenme: fskent; var enrnum: integer);
begin
  rtbcrx:=0;
  repeat
    rtbcrx:=rtbcrx+1;
    if rtb[rtbcrx].rnm = flenme
      then ftbcrx:=rtb[rtbcrx].ftbptr
  until (rtbcrx = rtbmx1) OR (rtb[rtbcrx].rnm = flenme);
  ftb[ftbcrx][enrnum]:=true
end.

```

Nous constatons qu'ici aussi, comme pour la procédure RCHNTB, il y a deux traitements à faire: un pour le premier enregistrement et une boucle pour les enregistrements suivants. Nous prendrons donc un test identique au précédent, c'est-à-dire, premièrement, une table avec un seul enregistrement et deuxièmement une table avec au moins deux enregistrements.

Ces tests nous ont permis d'introduire les corrections suivantes:

- La boucle repeat...until stk stkrx .nrx = 0 doit être remplacée par la boucle while stk stkrx .nrx ≠ 0. En effet, cette boucle ne peut être exécutée s'il n'y a qu'un seul enregistrement;
- Dans les deux cas, en fin de boucle while k ≠ nse+1, il faut tester s'il y a encore un descripteur de noeud qui suit le dernier descripteur de noeud traité. Si oui, dans le cas où ntb ntbrx .nxp ≠ 0, on doit incrémenter k normalement: k:=k+1. Sinon, on pose k:=nse+1 pour terminer le traitement de l'enregistrement en cours.

2.2.2. Constitution de la table partielle des noeuds.

(Sélection et tri par la procédure UPDFAP)

Plutôt que de tester la procédure UPDFAP indépendamment de son environnement, ce qui nous obligerait à écrire des procédures qui permettent de mettre en évidence les résultats de la procédure ; nous allons nous servir des procédures de chargement déjà testées.

Nous entrerons, par lecture à l'écran, une table de noeuds qui sera rechargée sur différents fichiers par l'intermédiaire de UPDFAP et RCHNTB, puis chargée à nouveau en mémoire par CHGNTB. Les différences avec la table escomptée permettront de détecter les erreurs de UPDFAP.

Dans le choix des tables qui nous ont servi de données test, nous nous sommes basés sur les différents chemins de l'algorithme à parcourir, déterminé par les instructions conditionnelles.

Rappelons donc le canevas de cet algorithme ainsi que son idée.

La procédure consiste à recopier les descripteurs de noeuds de NTB appartenant à un même fichier dans la table intermédiaire PNT.

Il s'agit donc d'un simple recopiage, sauf en ce qui concerne les pointeurs nxp et fap qui vont être modifiés.

Le texte qui nous intéresse est donc le suivant :

```
begin
  pntcrx:=0
  repeat
    pntkpx:=pntcrx ;
    pntcrx:=pntcrx+1;
```

```

    if ntb[ntbcrx].lnb = 0
(a)      then
            pnt[pntcrx].nxp:=pntcrx+1
(b)      else
            begin
                crtlev:=pnt[pntkpx].lnb;
                pnt[pntcrx].nxp:=pntcrx+1;
                if pnt[pntcrx].lnb = crtlev
(c)          then pnt[pntcrx].fap:=pnt[pntkpx].fap;
                if pnt[pntcrx].lnb = crtlev+1
(d)          then pnt[pntcrx].fap:=pntkpx;
                if pnt[pntcrx].lnb < crtlev
(e)          then
                    begin
                        pntidx:=pntkpx
                        while pnt[pntidx].lnb > crtlev do
                            pntidx:=pnt[pntidx].fap;
                        pnt[pntcrx].fap:=pntidx
                    end
                end
                ntbcrx:=ntb[ntbcrx].nxp
            until ntbcrx = 0;
            pnt[1].fap:=pntcrx
        end.

```

Les chemins à parcourir sont "a", "bc", "bd", et "be". Etant donné que ces chemins sont parcourus dans une boucle, ils peuvent l'être 0, 1 ou plusieurs fois, sauf le chemin a qui lui est parcouru une et une seule fois (un arbre ayant une racine et une seule).

Premier test : passage uniquement par a.

Un arbre n'ayant d'autre noeud que sa racine.
Ce premier test met en évidence une erreur sur le pointeur `nxp`, qui est mis à `pntcrn + 1`, alors qu'il n'y a pas de suivant et qu'il devrait donc prendre la valeur 0.
Cette remarque est valable aussi dans le chemin b, pour le dernier noeud de l'arbre. Il faut donc remplacer

```
pnt [pntcrx] .nxp:=pntcrx + 1
```

par

```
if ntb [ntbcrx] .nnp = 0
  then pnt [pntcrx] .nxp:=0
  else pnt [pntcrx] .nxp:=pntcrx + 1;
```

Second test : passage par bc.

Il faut que l'arbre contienne deux noeuds qui se suivent dans l'ordre dynastique et qui soient au même niveau

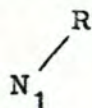
exemple



Troisième test : passage par bd.

Il faut un noeud qui soit de niveau d'unité supérieure au niveau de son précédent dans l'ordre dynastique.

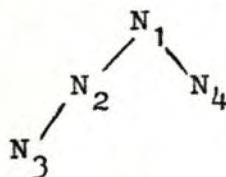
exemple



Il n'est pas absolument nécessaire de tester les chemins bc et bd, étant donné qu'il est évident que dans le premier cas, le noeud courant est le frère du précédent et que dans le second, ce noeud est le fils du précédent. Le

troisième chemin est lui plus complexe. Un test tout à fait général met en évidence l'erreur suivante.

Prenons l'exemple



Le niveau du noeud courant (N_4) est strictement inférieur au niveau du noeud précédent (N_3).

Pour cet exemple, la procédure donne comme résultat que N_4 est le père de N_3 , ce qui est faux.

La cause est facile à relever.

```
crtlev = pnt[pntkpx].lnb = pnt[pntidx].lnb
```

(dans l'exemple : pntkpx et pntidx désignent N_3)

crtlev = 2 = niveau de N_3)

et la condition de la boucle while n'est donc jamais vérifiée (en fait, n'importe quel test aurait révélé cette erreur).

Nous devons remonter le parcours dynastique depuis le noeud précédant le noeud courant, jusqu'à un noeud de niveau égal à ce noeud courant.

Le père du dernier noeud trouvé est le père du noeud courant. Il faut donc remplacer la séquence d'instruction concernée par :

```

while pnt[pntidx].lnb > pnt[pntcrx].lnb
do pntidx := pnt[pntidx].fab;
pnt[pntcrx].fab := pnt[pntidx].fab.

```

Un test reprenant toutes les possibilités, c'est-à-dire une combinaison des trois chemins, n'a pas révélé d'autres erreurs.

3. Procédures utilitaires pour le traitement des commandes.

Nous testerons dans ce chapitre TRTARG et TRTNID les procédures de traitement de l'argument et de l'identifiant de noeud d'une commande.

Ces procédures sont relativement complexes et il nous paraît préférable de les tester à part. De plus, TRTNID est utilisée par plusieurs autres procédures, il est donc préférable de la tester indépendamment de toutes ces procédures.

3.1. La procédure TRTARG.

Cette procédure traite l'argument d'une procédure de génération, c'est-à-dire qu'elle donne une valeur aux variables fstnbr et secnbr.

Un argument est défini par:

$\langle \text{argument} \rangle ::= \langle \text{empty} \rangle / \langle W \rangle / \langle W \rangle \langle \text{entier} \rangle / \langle W \rangle, \langle \text{entier} \rangle /$
 $\langle W \rangle \langle \text{entier} \rangle, \langle \text{entier} \rangle$

Cette définition nous donne explicitement les cas à tester. Soient donc les tests suivants avec les résultats attendus.

argument	fstnbr	secnbr
—	0	0
W	0	32767
W entier-1	entier-1	32767
W,entier-2	0	entier-2
W entier-1,entier-2	entier-1	entier-2

Mais avant de passer au test, il faut réécrire la fonction VALNBR. cette fonction transforme les entiers de l'argument lu sous forme d'une suite de caractères dans le fichier de commande, en une variable entière. Il suffit pour cela de transformer crtcar en entier, par exemple par la procédure MOVERIGHT (ou la fonction ORD) et de retrancher 48 à l'entier ainsi obtenu. La fonction peut alors s'écrire:

```
Function VALNBR: integer;

var oldvalnbr, nbr: integer;
begin
  oldvalnbr:= 0;
  MOVERIGHT (crtcar, oldvalnbr, 1);
  oldvalnbr:= oldvalnbr-48;
  GETSGNCAR (stk[stkcrx]);
  while (crtcar= '0') OR (crtcar= '1')
    OR (crtcar= '2')
    OR (crtcar= '3')
    OR (crtcar= '4')
    OR (crtcar= '5')
    OR (crtcar= '6')
    OR (crtcar= '7')
    OR (crtcar= '8')
    OR (crtcar= '9')
  do
    begin
      nbr:= 0;
      MOVERIGHT (crtcar, nbr, 1);
      nbr:= nbr-48;
      oldvalnbr:= oldvalnbr*10+nbr;
      GETSGNCAR (stk[stkcrx])
    end
  valnbr:= oldvalnbr
end.
```


Remarque.

Nous n'avons pas utilisé la déclaration SET OF '0'..'9' pour réduire le texte, un test effectué révélant que certains chiffres appartenaient à ce set et d'autres pas.

Revenons au test proprement dit.

La procédure qui doit servir de support au test de TRTARG doit disposer de certaines déclarations globales, par exemple GETSGNCAR,...

Nous lui adjoindrons la librairie DECL qui contient ces déclarations.

Au point de vue des traitements, on se contente de lire à l'écran, par exemple, un argument, de le ranger dans `stk[stkcrx].buf.idt`, de faire un appel à TRTARG et d'afficher les résultats `fstnbr` et `secnbr`.

Ces résultats se sont révélés corrects, sauf dans le cas W, entier-2, où `fstnbr` prend une valeur erronée.

Pour corriger cette erreur, nous introduisons la procédure ABSFSTNBR qui doit s'exécuter quand, dans l'état 1, on rencontre une virgule.

L'entrée [1,2] de la table ACTARC prendra la valeur 12 et non zéro, ce qui impliquera l'exécution de la procédure. ABSFSTNBR aura pour simple effet de donner la valeur 0 à `fstnbr`.

D'autres données avec des arguments incorrects nous ont permis de tester le traitement syntaxique.

Exemple: omission de W, de la virgule, entiers incorrects.

On remarque ainsi que rien n'est prévu si un des entiers de l'argument dépasse `maxint`. Si ce cas se présente, il y aura un dépassement de capacité dans `VALNBR` et cela peut se passer à cause d'un oubli de la virgule.

W47,275 écrit W47 275 est interprété comme W47275 et fstnbr dépasse la capacité sans que le système puisse avertir l'utilisateur de la cause de l'erreur et en interdisant toute reprise.

Nous pourrions corriger cela en déclarant oldvalnbr comme "long integer", ce qui est permis sur Apple, et en vérifiant qu'il ne dépasse jamais maxint.

Nous avons testé et corrigé la procédure de traitement d'un argument d'une commande en fin du mémoire dans le texte de toute l'application.

3.2. La procédure TRTNID.

Cette procédure traite l'identifiant de noeud d'une commande et donne comme résultat les indices, dans la table des noeuds, des noeuds identifiés. Un identifiant de noeud est défini de la manière suivante:

$\langle \text{identifiant de noeud} \rangle ::= \{ \langle \text{nom de noeud} \rangle \}_0^1 \langle \text{queue d'identifiant} \rangle$

$\langle \text{queue d'identifiant} \rangle ::= \langle \text{séparateur} \rangle \langle \text{nom de noeud} \rangle / \langle \text{queue d'identifiant} \rangle \langle \text{séparateur} \rangle \langle \text{nom de noeud} \rangle$

$\langle \text{séparateur} \rangle ::= */. \{ \}$ tels que

toute combinaison *,. est exclue

* apparaît uniquement en tête d'un identifiant

La variable de test n'est ici pas uniquement l'identifiant de noeud mais également la table des noeuds sur laquelle nous allons travailler.

La table devrait être la plus générale possible et devrait comporter: plusieurs arbres

plusieurs niveaux dans chaque arbre



Nous donnerons ci-dessous le texte de ANAWAY et SCHNDS, les deux outils principaux de TRTNID utilisés pour: retrouver un noeud dans la table (SCHNDS)

analyser le chemin qui le relie à la racine et le comparer avec le chemin défini par l'identifiant (ANAWAY)

Ce texte nous permettra de situer plus facilement les corrections à effectuer.

Procédure SCHNDS;

```

begin
  ntbcx:= 1;
  cmnkpx:= cmncrx;
  while ntbcx ≤ ntbmxi do
    begin
      if ntb[ntbcx].nnm ≠ nsk[cmnkpx]
      then ntbcx:= ntbcx+1
      else
        begin
          ntbkpx:= ntbcx;
          ANAWAY
        end
      end
    end
  end,

```

Procedure ANAWAY;

var idnnnm: boolean;

i: integer;

begin

idnnnm:= true;

while (cmncrx > 1) AND (idnnnm = true) do begin

for i:= 1 to lsk[cmncrx] do

ntbcrx:= ntb[ntbcrx].fap;

cmncrx:= cmncrx-1;

if ntb[ntbcrx].nnm \neq nsk[cmncrx]

then idnnnm:= true;

if idnnnm = true

then if (lsk[cmncrx] = -1 OR lsk[cmncrx] =
ntb[ntbcrx].lnb)

then UPDASK

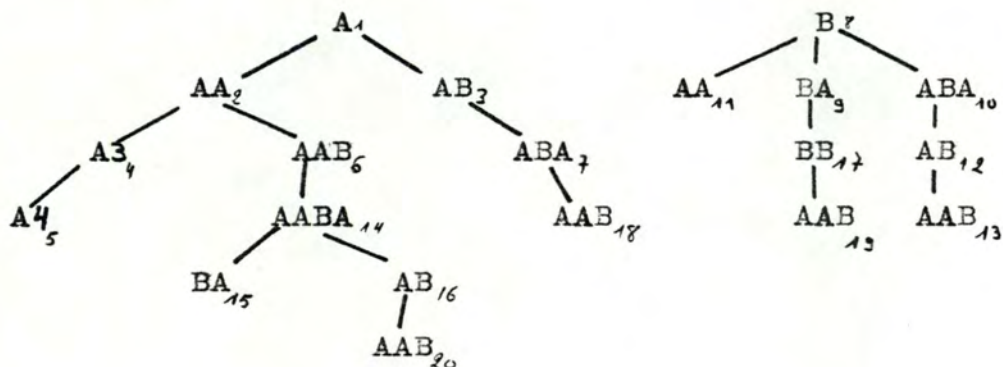
end;

ntbcrx:= ntbkpx+1;

cmncrx:= cmnkpx

end.

Voici les deux arbres constituant une table de noeuds sur lesquels ont portés les tests



Les nombres en indice sont les indices de la table des noeuds. Pour chaque test effectué, nous avons donné le résultat attendu, le résultat obtenu et l'explication de l'erreur, s'il y en a une.

identifiant	résultat attendu	résultat obtenu
A....BA	15	9, 15
A...AAB	18	6, 18
A....AB	16	3, 12, 16
A.....AAB	20	6, 13, 18, 19, 20
A.....AABA...AAB	20	20
A.....AB.AAB	20	13, 20

On obtient les noeuds de noms voulus, avec les noeuds intermédiaires corrects. Sont incorrects par rapport à l'identifiant: le numéro de niveau et la racine.

Pourquoi?

On ne teste pas quand, en remontant de père en père, on arrive à la racine. Or cette racine a aussi un "père", qui est en fait l'indice de l'entrée correspondante dans la table des racines. Mais cet indice peut être celui d'un noeud de la table des noeuds.

Prenons l'exemple de A....AB, qui doit donner 16 et qui donne 3, 12 et 16. Le noeud 3 est le fils de A. Remontons 4 étages depuis ce noeud. Le premier nous amène à A, dont le pointeur fap vaut 1 (puisque 1ère racine). Comme A est aussi le noeud d'indice 1 dans la table, on va monter les quatre étages en restant en A.

Le test de correspondance

ntb ntbcx .nm = nsk cmncx

va être positif et le noeud sera accepté!

Le noeud 12 est petit-fils de B, la deuxième racine. En remontant deux étages, on arrive à B, dont le pointeur fap vaut 2. Un étage supplémentaire nous mène au deuxième noeud de la table, c'est-à-dire AA, fils de A. Le quatrième étage nous fait remonter à A et de nouveau, le noeud peut être accepté.

Correction:

Il ne faut jamais remonter au-dessus d'une racine. Pour cela, chaque fois que l'on remonte d'un étage, il faut tester si on n'est pas arrivé à une racine. Si c'est le cas et qu'on n'a pas franchi le nombre d'étages voulu, alors le noeud considéré est à rejeter.

identifiant	résultat attendu	résultat obtenu
AB..AAB	18	—
AA..A4	5	—

Des noeuds existants ne sont pas trouvés.

Pourquoi?

La pile lsk contient les numéros de niveau des noeuds successifs de l'identifiant. Etant donné que l'identifiant

commence par un nom, on exécute la procédure INTTRTNM qui fait $lsk[cmncrx] := levnbr$.

A ce moment, $cmncrx = 1$ et $levnbr = 0$. Or le numéro de niveau des noeuds dont le nom commence l'identifiant n'est pas 0 puisque ce ne sont pas des racines. Comme en fin de procédure ANAWAY, on fait UPDASK si

$lsk\ cmncrx = ntb\ ntbcx.lnb$,
et que dans ce cas, la condition n'est pas vérifiée,
($ntb[ntbcx].lnb = 1$ et $lsk[cmncrx] = 0$),
le noeud n'est pas mis dans la pile des résultats.

Correction:

Il faut initialiser la pile lsk, dans le cas où on commence l'identifiant par un nom, par le numéro de niveau du noeud représenté par ce nom. Or il peut y en avoir plusieurs comme c'est le cas pour AB. Il faudrait donc recommencer pour chaque noeud, en réinitialisant $lsk[1]$.

Nous n'avons pas fait cette correction car, d'une part, elle aurait demandé de trop grandes modifications dans la logique de la procédure et, d'autre part, on peut remplacer AB...AAB par *AB...AAB sans rien perdre des possibilités de la procédure

identifiant	résultat attendu	résultat obtenu
A...AABA...AAB	20	—
B.BA...AAB	19	—

Le nom de noeud existe, le nombre de niveaux qui le relie à la racine est correct, le chemin est correct; pourtant,

les noeuds ne sont pas identifiés.

Pourquoi?

La procédure ANAWAY n'analyse pas correctement le chemin à parcourir du dernier noeud au premier de l'identifiant.

On remonte ce chemin par l'instruction

```
for i:= 1 to lsk[cmncrx] do  
  ntbcx:= ntb[ntbcx].fap
```

Or lsk[cmncrx] contient, au départ, le niveau du dernier noeud de l'identifiant. Cette boucle fait donc remonter directement jusqu'à la racine.

Le test `if ntb[ntbcx].nnm ≠ nsk cmncrx` est donc toujours positif s'il y a un noeud intermédiaire et le chemin est considéré incorrect et est refusé.

Correction:

Il faut remonter du nombre d'étages qui séparent un noeud du noeud qui le précède dans l'identifiant, ce qu'on peut faire par l'instruction

```
for i:= 1 to (lsk[cmncrx]-lsk[cmncrx-1]) do  
  ntbcx:= ntb[ntbcx].fap;
```

identifiant	résultat attendu	résultat obtenu
*AB.AAB	20, 13	18
*AAB...AAB	20	—
*AB..AAB	18	—
*AA.AAB	6	—

On remarque que le premier test fournit le résultat attendu pour le troisième alors que les identifiants ne

diffèrent que par le nombre de niveaux intermédiaires.

Pourquoi?

C'est la procédure ANAWAY qui ne compte pas le nombre de niveaux correctement entre chaque noeud.

Rappelons qu'une correction précédente nous a fait écrire

```
for i:= 1 to (lsk[cmncrx]-lsk[cmncrx-1]) do  
  ntbcx:= ntb[ntbcx].fap;
```

Or, dans ce cas-ci, nous avons $lsk[1] = -1$, à cause de "*" qui précède l'identifiant.

le dernier écart est donc calculé à partir de -1 alors qu'il devrait l'être à partir de 0.

Correction:

```
if lsk[cmncrx-1] = -1  
  then ecart = lsk[cmncrx]  
  else ecart = lsk[cmncrx]-lsk[cmncrx-1];  
for i:= 1 to ecart do  
  ntbcx:= ntb[ntbcx].fap;
```

identifiant	résultat attendu	résultat obtenu
..AB	12	—
...AAB	13, 18, 19	—
*AB	3, 12, 16	—

Les noeuds sont complètement ignorés. on remarque que dans ANAWAY, rien n'est fait si, en entrée, $cmncrx = 1$, ce qui arrive dans les cas cités: il n'y a qu'un seul nom de noeud dans l'identifiant.

Pourquoi?

Cela est du à la forme de la boucle

```
while (cmncrx > 1) AND (idnnnm = true)
qui ne s'exécute jamais si cmncrx = 1
```

Correction:

Il suffit d'introduire un test supplémentaire. on peut cependant distinguer deux cas:

1er cas *AB
.....

Tous les noeuds AB trouvés dans la table par SCHNDS doivent être acceptés; il est inutile d'appeler ANAWAY pour ceux-là. on va donc introduire la suite d'instructions suivantes dans SCHNDS:

```
if (cmncrx = 1) AND (lsk[1] = -1)
  then
    begin
      UPDASK;
      ntbcx:= ntbcx+1
    end
  else
    ANAWAY
```

2ème cas ..AB
.....

Il faut analyser le chemin et donc appeler ANAWAY. Dans cette procédure, on va ajouter les instructions

```
if cmncrx = 1
  then
    begin
      i:=0;
      repeat
        ntbcx:= ntbcx[ntbcx].fap;
        i:= i+1
      until ntbcx[ntbcx].lnb = 0;
      if i = lsk[cmncrx] then UPDASK
```


Pour chaque noeud de la table, trouvé par SCHNDS, il suffit de compter le nombre de niveaux qui le sépare de la racine et de le comparer au numéro de niveau du noeud de l'identifiant.

Conclusion.

Nous avons corrigé toutes les erreurs que nous avons détectées dans TRTNID, à l'exception d'une seule.

L'identifiant A.. .B ne donne rien si A n'est pas une racine. Il faut, dans ce cas, utiliser *A.. .B.

Nous n'avons pas testé ici le cas d'un noeud inexistant, ou le cas d'un identifiant de noeud donnant plusieurs noeuds, alors qu'il est précédé de "!".

La variable nmb, qui compte les noeuds identifiés, est en effet testée en dehors de TRTNID, par les procédures appelantes.

4. Procédures de traitement des commandes.

Les procédures que nous devons tester sont les suivantes:

TRTCRE: introduction d'un noeud dans la table, en dernière position dans l'ordre dynastique, avec éventuellement création de son contenu.

TRTGEN: génération du contenu d'un ou de plusieurs noeuds.

TRTINS: insertion d'un noeud dans la table.

TRTMOD: modification du contenu d'un ou de plusieurs noeuds.

TRTADD: addition au contenu d'un ou de plusieurs noeuds.

TRTDEL: suppression d'un sous-arbre de la table des noeuds.

Toutes ces procédures fonctionnent suivant le même schéma:

- lecture d'une commande dans le fichier de commande

- traitement de la commande:

- . mise à jour de la table des noeuds
- . mise à jour des contenus de noeuds sur la disquette

Pour tester ces procédures, il nous faut un moyen pour contrôler leurs effets. Ce moyen n'est pas prévu dans l'application. Nous allons écrire un programme, extérieur à l'application, qui permet d'avoir accès aux données contenues dans les fichiers de l'application, c'est-à-dire les tables des noeuds de ces fichiers et les contenus des noeuds.

Pour construire ce programme, nous pouvons utiliser les procédures qui ont été prévues dans l'application et qui nous intéressent ici, c'est-à-dire les procédures

de chargement de table (CHGNTB) et la lecture des enregistrements d'un fichier (READFILE).

Nous ne donnerons ici que les spécifications du programme, Le texte Pascal complet se trouvant en fin de l'ouvrage.

Le programme TEST est un programme qui affiche, à l'écran ou à l'imprimante, la table des noeuds, le contenu de chaque noeud, ou les deux suivant le choix de l'utilisateur.

4.1. La procédure TRTCRE.

Nous devons tester

- la création d'une racine
- la création d'un noeud quelconque
- la création d'un noeud avec un contenu sans génération

La création du contenu d'un noeud avec génération utilise la procédure TRTGEN. Nous testerons ce cas dans le chapitre suivant.

Nous testerons aussi, dans ce chapitre, la procédure TRTEND, dont l'utilisation est liée à celle de TRTCRE.

Le premier test que nous avons effectué porte simplement sur la création de noeuds et de racines, sans contenu. Le fichier de commandes fourni au système contenait donc des commandes CREATE entrecoupées de commandes END, ces dernières étant placées de telle sorte que certaines d'entre elles terminaient une structure arborescente et entraînaient la création d'une racine.

On a aussi testé le cas de reprise d'une structure arborescente.

Les erreurs qui ont ainsi été détectées n'ont pas posé

de problème pour la correction:

- Dans TRTSCHLND
.....

Recherche du dernier noeud d'une structure arborescente, en vue d'une reprise.

- . possibilité d'une erreur de syntaxe: ")" absente après le nom du fichier.
- . absence de la mise à jour de crtlev, qui doit se faire par l'instruction: crtlev:= ntb[lnd].lnb

- Dans SETCRN
.....

- . possibilité d'une erreur de syntaxe: absence de "" ou de ";" après le dernier caractère du nom du noeud.

- Dans TRTNOD
.....

- . erreur dans la mise à jour du pointeur fap du noeud créé:

on a ntb[ntbcrx].fap:= ntb[ntbidx].fap
où ntbcrx désigne le noeud créé

ntbidx désigne le noeud obtenu en remontant à partir du dernier noeud de la structure arborescente jusqu'au moment où le numéro de niveau soit égal ou inférieur à celui du noeud créé:

```
ntbidx:= lnd;  
while ntb[ntbidx].lnb > crtlev do  
    ntbidx:= ntb[ntbidx].fap
```

Il se peut que le noeud qu'on veut créer soit le fils du dernier noeud créé. Dans ce cas,

ntb[ntbidx].lnb = oldlev-1,

et on ne passe pas dans la boucle. Il faut alors que

ntb[ntbcrx].fap:= ntbidx

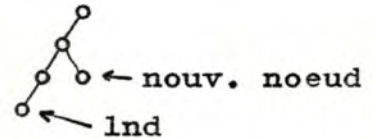
La mise à jour du pointeur se fait alors par


```

if ntb[ntbidx].lnb = oldlev
  then ntb[ntbcrx].fap:= ntb[ntbidx].fap

```

Ce cas correspond au schéma

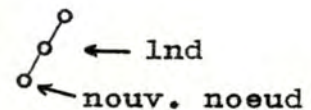


```

if ntb[ntbidx].lnb = oldlev-1
  then ntb[ntbcrx].fap:= ntbidx

```

Ce cas correspond au schéma



Le deuxième test effectué porte sur la création d'un noeud avec contenu, sans génération.

(ette création se base essentiellement sur les procédures TRTCONNOD et TRTENDCRE.

La seule distinction que l'on peut apporter dans ces tests, c'est de donner un contenu qui tient sur une ligne de commande ou bien un contenu réparti sur plusieurs lignes de commandes; de même, un contenu tenant dans un seul enregistrement du fichier ou bien réparti sur plusieurs enregistrements.

Nous avons pris des contenus de longueur variable, à l'intérieur des différentes catégories déjà citées.

Nous avons ainsi remarqué que certains contenus étaient "parasités" par des morceaux de contenus de noeuds précédents. Cela venait du fait que le buffer de sortie (outrec) n'était pas nettoyé après chaque écriture d'un enregistrement. Ce nettoyage a été ajouté dans la procédure TRTENDCRE.

Nous avons aussi testé la procédure TRTCONNODVAL grâce à la commande CV qui effectue le même traitement que TRTCONNOD, dans le cas où il n'y a pas de commande de génération dans le contenu du noeud créé.

La création avec génération, qui utilise TRTCRE (avec

l'option V pour Value) et TRTCONNODVAL, sera testée dans le chapitre suivant.

4.2. La procédure TRTGEN.

Cette procédure traite la génération du contenu d'un ou de plusieurs noeuds dans le contenu d'un autre noeud. Elle est utilisée au moment de la création d'un noeud, quand, dans le contenu du noeud créé, on rencontre une commande de génération. Elle est alors appelée par la procédure TRTCONNODVAL de TRTCRE.

Les procédures utilisées pour le traitement sont TRTARG, TRTNID qui ont déjà été testées et dont le fonctionnement correct est assuré; GEN, qui parcourt tous les noeuds identifiés par TRTNID et TRTARG; et GENREC, qui génère le contenu d'un des noeuds identifiés. Les procédures à tester sont donc TRTCONNODVAL, GEN et GENREC.

La procédure GEN détermine les noeuds dont le contenu doit être généré. Elle sera donc testée en faisant varier l'argument et l'identifiant de noeud de la commande de génération. Rappelons que l'argument détermine quels noeuds du sous-arbre, ayant le noeud identifié pour racine, seront pris en compte; les deux entiers de l'argument donnant un intervalle de niveaux relatif à la racine du sous-arbre.

En donnant tous les cas possibles d'argument et en les combinant avec différents cas d'identifiants, on a pu détecter les erreurs suivantes dans GEN:

- dépassement de la capacité, du au test

```
if ntb ntbcx .lnb oldlev+secnbr
```

car secnbr peut prendre la valeur maxint.

- valeur d'indice illégale de ntb à cause de

```
ntbcx:= ntb ntbcx .npx
```

Le passage au noeud suivant du sous-arbre est impossible
s'il n'y a pas de noeud suivant (npx = 0)

Nous avons corrigé cela de la manière suivante:

```
var bool: boolean;
```

```
⋮
```

```
repeat
```

```
if {secnbr = maxint)
```

```
OR ((secnbr < maxint) AND (ntb[ntbcx].lnb ≤ oldlev  
+maxint))
```

```
then
```

```
if ntb[ntbcx].lnb ≥ oldlev+fstnbr
```

```
then
```

```
GENREC (crtadr);
```

```
ntbcx:= ntb[ntbcx].npx;
```

```
if ntbcx = 0 then bool:= true
```

```
else if ntb[ntbcx].lnb = oldlev
```

```
then bool:= true
```

```
else bool:= false
```

```
until bool;
```

La procédure TRTGEN est une procédure récursive, Nous avons donc introduit des commandes de création avec aucun, un ou plusieurs appels récursifs. Nous avons ainsi pu détecter des erreurs dans le sauvetage des variables.

L'appel récursif, ainsi que les sauvetages, se font dans GENREC. Nous avons deux piles de sauvetage:

- stk, pour les enregistrements successifs

- ssk, pour les variables du système (ntbcx et askkpx)

1° Considérons d'abord le stack stk. Dès qu'on entre dans GENREC, on sauve stk par

```
stk[stkcrx].crp:= idtcx;  
et stkcrx:= stkcrx+1
```

puisqu'il va être occupé par le contenu du noeud qu'on doit générer. S'il n'y a pas d'appel récursif, on sort de GENREC sans appel à TRTGEN, sans passer par la restauration du stack.

Il faut corriger cela en plaçant cette restauration à la fin de la procédure GENREC:

```
stkcrx:= stkcrx-1;  
et idtcx:= stk[stkcrx].crp
```

2° D'autre part, tout appel de la procédure TRTGEN doit être précédé d'un sauvetage de la pile ssk par sskcrx:= sskcrx+1.

La procédure TRTCONNODVAL, qui avait déjà été testée, mais sans appel à TRTGEN, n'a pas du être modifiée car son traitement (copiage du contenu de la commande dans l'enregistrement de sortie) n'est interrompu que par TRTGEN, et les variables, par exemple idtcx, sont correctement restaurées.

Cependant, on remarque que, en sortant de TRTGEN, TRTCONNODVAL saute au caractère suivant, qui peut être un blanc. Mais dans TRTGEN, TRTNID a déjà sauté au caractère non blanc qui suit l'identifiant de noeuds: on perd ainsi tous les caractères blancs et un caractère non blanc après cet identifiant.

Considérons l'exemple suivant:

```
/C, N1"Ceci est le contenu du noeud N1  
/G (*N2) bbbbbb ceci est la suite de ce contenu."
```


TRTNID se termine nécessairement, de part sa construction, par GETSGNCAR, et positionne alors crtcar sur "C". On perd les caractères blancs qu'on aurait voulu insérer à cette place.

On peut corriger cet effet en plaçant un caractère non blanc juste derrière la parenthèse de l'identifiant.

Il faut donc faire suivre un identifiant de noeud par un caractère non blanc quelconque.

4.3. La procédure TRTADD.

La procédure TRTADD ajoute le contenu présent dans le texte de la commande au contenu de tous les noeuds donnés par l'identifiant de noeud.

Pour chaque noeud donné, il faut chercher le dernier enregistrement du contenu de ce noeud et le compléter avec le début du contenu de la commande, en ajoutant éventuellement des enregistrements, si nécessaire (procédure ADDCON). Les tests devront porter sur:

- 1° la longueur du contenu à ajouter (une ou plusieurs lignes de commande, un ou plusieurs enregistrements)
- 2° le nombre de noeuds identifiés (un ou plusieurs)
- 3° la longueur du contenu déjà existant (aucun ou plusieurs enregistrements)

1er test: un seul noeud avec contenu
.....

un contenu à ajouter sur une seule ligne
de commande

On obtient dans le dernier enregistrement du noeud des morceaux de son ancien contenu, le contenu à ajouter ayant disparu. Celui-ci se trouve en stk[1].buf.idt et

commence en idtcx.

Or, les enregistrements successifs du contenu du noeud sont lus sur stk[1].buf.idt, en écrasant le contenu à ajouter et en modifiant la valeur de idtcx.

Correction.

- Il faut sauver idtcx avant la lecture du contenu existant et restaurer idtcx avant l'appel de TRTCONNOD.
- Il faut lire le contenu existant sur un autre étage du stack, par exemple stk[2].

Mais cela ne suffit pas. En effet TRTCONNOD va remplir OUTREC, le record de sortie, avec le contenu à ajouter. Mais OUTREC doit d'abord être initialisé avec la fin du contenu existant.

D'autre part, il faut que le dernier enregistrement, incomplet, soit remplacé par l'enregistrement obtenu en le complétant. On peut, par exemple, le réécrire; mais pour cela, il faut retenir son adresse au moment où il est lu sur le stack.

2ème test: plusieurs noeuds avec ou sans contenu
.....
un contenu à ajouter sur une seule ligne
de commande

Le contenu est ajouté au premier noeud mais pas aux suivants.

Explication.

En lisant le contenu à ajouter, TRTCONNOD déplace le pointeur idtcx vers la fin de la ligne de commande. Ce pointeur n'est pas restauré pour le noeud suivant et le traitement ne peut se faire.

Correction.

Avant tout traitement, on sauve la position du début du

contenu à ajouter: stk[1].crp:= idtcx.

Le sauvetage avant la lecture du contenu existant devient inutile. Il faut restaurer idtcx avant tout appel à TRTCONNOD, de même que crtcar qui a, lui aussi, été modifié: crtcar:=stk[1].buf.idt[idtcx].

Dans le cas où le noeud n'avait pas de contenu, son nouveau contenu est parasité par du texte indésirable.

Comme pour le premier cas, il faut initialiser le record de sortie avec la fin du texte précédent, c'est-à-dire à blanc, et surtout, positionner son indice au début: odtcx:= 0.

3ème test: un seul noeud avec ou sans contenu
.....
 un contenu à ajouter sur plusieurs lignes
 de commande

Il n'y a pas eu de correction à faire car la procédure TRTCONNOD, principale responsable de ce traitement, a déjà été testée.

4ème test: plusieurs noeuds, avec ou sans contenu
.....
 un contenu à ajouter sur plusieurs lignes
 de commande

Seul le premier noeud a été correctement traité.

Explication.

Après le traitement du premier noeud, on dispose, sur le stack, de la dernière ligne de la commande correspondant à la fin du contenu. Les premières lignes, avec le début à ajouter, sont perdues.

Correction.

Il y a deux manières de corriger cela:

- Soit retenir la position, dans le fichier, du premier caractère du contenu et y revenir par la procédure SEEK. Il faudrait pour cela, compter chaque caractère du fichier au fur et à mesure de leur lecture.
- Soit sauver successivement, sur le stack, toutes les lignes d'une même commande.

C'est la procédure TRTCONNOD qui devrait rechercher sur le stack les lignes de commandes successives.

Le sauvetage devrait se faire en dehors de TRTCONNOD puisque cette procédure est exécutée pour chaque noeud traité. Mais comme c'est elle qui lit le contenu de la commande, ce n'est pas possible. Il faudrait alors une procédure, extérieure à TRTCONNOD, qui lise la commande et mette ses lignes sur le stack à la disposition de TRTCONNOD.

Aucune de ces deux solutions n'a été implémentée jusqu'à présent.

Nous donnerons en conclusion le texte final de la procédure ADDCON

```
var fstidx : 0 . . ntbmx1;
    fstlev : integer;

begin
  stk[1].crp:= idtcx;    sauvetage de idtcx
repeat
  askcrx:= askcrx-1;
  fstidx:= ask[askcrx];
  fstlev:=ntb[fstidx].lnb;
  if ntb[fstidx].rep = 0
  then
    begin
      rnb:=0;
```



```

idterx:=stk[1].crp;
crtcar:=stk[stkerx].buf.idt[idterx];
for odterx:=1 to odtmxl
    do outrec.odt[odterx]:=' ';
odterx:=0;
TRTCONNOD(fstidx)
end
else
begin
    if fstlev = 0
        then for nnmcx:=1 to nnmmxl
            do stk[2].fnm[nnmcx]:=
                ntb[fstidx].nnm[nnmcx]
        else for nnmcx:=1 to nnmmxl
            do stk[2].fnm[nnmcx]:=
                ntb[ntb[fstidx].rtp].nnm[nnmcx] ;
stk[2].buf.nxr:=ntb[fstidx].rcp;
rnb:=0;
repeat
    crtngx:=stk[2].buf.nxr;
    RDLIN(stk[2]);
    rnb:=rnb+1
until stk[2].buf.nxr = 0;
odterx:=1;
while stk[2].buf.idt[odterx] ≠ chr(3)
do
    begin
        outrec.odt[odterx]:=stk[2].buf.idt[odterx];
        odterx:=odterx+1
    end;
odterx:=odterx-1;
idterx:=stk[1].crp;
crtcar:=stk[stkerx].buf.idt[idterx];
TRTCONNOD(fstidx)
end
until askerx = askkpx
end.

```

4.4. La procédure TRTINS.

La procédure TRTINS crée un noeud avec ou sans contenu et l'insère dans la table des noeuds à une place déterminée par l'utilisateur:

- soit comme fils aîné d'un noeud existant (procédure TRTEL)
- soit comme fils cadet d'un noeud existant (procédure TRTYO)
- soit comme frère cadet d'un noeud existant (procédure TRTYB)

Ce noeud existant sera donné par la partie "identifiant de noeud" de la commande. Ce noeud peut donc ne pas être identifié de manière unique: un nouveau noeud peut être inséré en plusieurs endroits de la table. Il y a deux parties dans le traitement de l'insertion du nouveau noeud: - l'insertion du noeud

- la création de son contenu

Comme nous avons la possibilité de créer plusieurs fois le même contenu, nous aurons les mêmes problèmes que ceux que nous avons rencontrés dans la procédure TRTADD. Si le contenu tient sur une seule ligne de commande, il faudra restaurer la bonne valeur de idtcx avant chaque appel de TRTCONNOD ou TRTCONNODVAL.

La correction à faire pour le cas où le contenu se répartit sur plusieurs lignes est celle présentée pour TRTADD et qui n'a pas été implémentée.

Vu cette dernière remarque et le fait que TRTC TRTCONNOD et TRTCONNODVAL ont déjà été testées et corrigées, nous nous contenterons de considérer le test de l'insertion d'un noeud dans la table.

Les tests que nous devons faire portent sur les trois procédures d'insertion dans une situation particulière de la table des noeuds:

Insertion du fils aîné d'un noeud

- qui n'a pas encore de fils et a un frère cadet
- qui n'a pas encore de fils ni de frère cadet
- qui a un fils
- qui a plusieurs fils

Insertion du fils cadet d'un noeud

- qui n'a pas de fils
- qui a un fils
- qui a plusieurs fils
- dont le fils cadet a des descendants
- dont le fils cadet n'a pas de descendant
- qui a un frère cadet
- qui n'a pas de frère cadet

Insertion du frère cadet d'un noeud

- qui a déjà un frère cadet
- qui n'a pas de frère cadet
- qui a des descendants
- qui n'a pas de descendant

Tous ces tests ont pour objectif de vérifier que l'ordre dynastique est bien respecté.

Dans le cas de l'insertion du frère cadet d'un noeud qui n'avait pas de descendant, l'indice `lstidx` du noeud précédant le nouveau noeud dans la chaîne dynastique n'était pas mise à jour. On corrige cela par:

```
if ntb[ntbcrx].lnb < fstlev then lstidx:= fstidx
```

où `fstidx` désigne l'indice du noeud dont on veut créer le frère

`ntbcrx = ntb[fstidx].nxp`, désigne l'indice du suivant de ce noeud

`fstlev` désigne le niveau de ce noeud

`lstidx` désigne l'indice du noeud précédant le noeud à créer dans la chaîne hiérarchique

La même erreur se reproduit dans TRTYO pour la création d'un fils cadet si le noeud dont on veut créer le fils n'en a pas encore. La correction se fait de la même manière par:

```
if ntb[ntbcrx].lnb < fstlev then lstidx:= fstidx
```

Dans la procédure TRTEL, où le noeud créé est toujours le suivant immédiat du noeud dont on veut créer le fils aîné, il n'y a pas de problème. On peut noter que dans le deuxième cas, conduisant à un résultat erroné, la procédure utilisée n'était pas la mieux adaptée. Pour donner un fils à un noeud qui n'en a pas, il vaut mieux créer un fils aîné qu'un fils cadet. Mais, à moins de signaler explicitement ce fait à l'utilisateur, il vaut mieux effectuer la correction pour que la procédure fonctionne dans tous les cas.

4.5. Procédure TRTMOD.

Cette procédure permet de remplacer soit le nom, soit le contenu d'un ou de plusieurs noeuds donnés par l'identifiant. Elle utilise deux procédures:

- MODNNM pour le remplacement du nom,
- MODCON pour le remplacement du contenu.

MODNNM ne pose pas de problème, elle ne fait que changer une variable en mémoire centrale. tandis que pour la procédure MODCON se pose de nouveau les mêmes problèmes que pour TRTADD:

- écrasement du contenu de la commande, corrigé par la lecture de l'ancien contenu dans une autre entrée du
- nécessité du sauvetage et de la restauration de idtcrx et de crtcar, s'il y a plusieurs noeuds à modifier

- impossibilité de modifier le contenu de plusieurs noeuds si le nouveau contenu est réparti sur plusieurs lignes de commande.

4.6. Procédure TRTDEL.

Cette procédure permet d'effacer tous les sous-arbres dont les racines sont les noeuds donnés par l'identifiant. Elle utilise pour cela:

- DELALL qui supprime tous les noeuds d'un arbre
- DELNDS qui supprime tous les noeuds d'un sous-arbre

Nous avons testé trois cas:

- 1° la suppression d'un arbre entier
- 2° la suppression d'un noeud quelconque
- 3° la suppression d'un noeud tel qu'un de ses descendants soit le dernier noeud de la structure arborescente

Le premier test n'a pas révélé d'erreur. Le deuxième a montré que le pointeur nxp (vers le suivant) du noeud précédant celui qui est supprimé n'est pas à jour, mais on peut l'y mettre par la séquence d'instructions suivantes:

```
ntbcrx:= ntb[fstidx].fap;    on remonte au père du noeud
                             à supprimer
repeat
  ntbkpx:= ntbcrx;
  ntbcrx:= ntb[ntbcrx].nxp    on suit le parcours dynasti-
                             que jusqu'à ce qu'on arrive
                             au noeud à supprimer
until[ntbcrx]= fstidx;
ntb[ntbkpx].nxp:= ntb[1stidx].nxp
                             ntbkpx retient l'indice du
                             noeud précédent
```

Le troisième test a montré que ce cas n'était pas prévu. Quand on parcourt le sous-arbre à supprimer, il faut tester en chaque noeud si celui-ci n'est pas le dernier. Si oui, on s'arrête; sinon, on va au noeud suivant.

ANNEXE UN

LE TEXTE DU PROGRAMME

(*S+*)

program sgfa(input,output);

uses (*\$U APPLE2:DECLIB.CODE *)decl,
(*\$U APPLE2:INTERLIB.CODE *)inter,
(*\$U APPLE2:CONTLIB.CODE *)contenu;

var reponse:char;
ix,jx:integer;

(*I APPLE2:CIMAD.TEXT *)

begin(*PROGRAM TESTCORE*)

trtint;

while not eof(f0)

do

begin

rdlin(stk[stkcrx]);

idtcx:=0;

getsgncar(stk[stkcrx]);

if crtcar='/'

then

getsgncar(stk[stkcrx])

else

begin

numerr:=1;

erreur(numerr)

end;

if (crtcar <> 'C')

and (crtcar <> 'E')

and (crtcar <> 'I')

and (crtcar <> 'D')

and (crtcar <> 'A')

and (crtcar <> 'M')

then

begin

numerr:=1;

erreur(numerr)

end;

case crtcar of 'C':begin

getsgncar(stk[stkcrx]);

trtcre

end;

'E':begin

getsgncar(stk[stkcrx]);

trtend

end;

'I':begin


```

        getsnrcar(stk[stkcrx]);
        trtins
    end;
'D':begin
    getsnrcar(stk[stkcrx]);
    trtdel
end;
'M':begin
    getsnrcar(stk[stkcrx]);
    trtmod
end;
'A':begin
    getsnrcar(stk[stkcrx]);
    trtadd
end

```

```

end;

```

```

    get(f0)
end;

```

```

trttr

```

```

end. (*SGFA*)

```

procedure tstrotcmp;

var blanc:packed array [1..nnmmx1] of char;

begin

for i:=1 to nnmmx1
do blanc[i]:=' ';

rtbcrx:=0;

repeat rtbcrx:=rtbcrx+1

until (rtb[rtbcrx].rnm=crn) or (rtb[rtbcrx].rnm=blanc) or (rtbcrx=rtbmx1);

if rtb[rtbcrx].rnm=blanc

then rtbkpx:=rtbcrx

else begin

if rtbcrx=rtbmx1 then numerr:=3;

if rtb[rtbcrx].rnm=crn then numerr:=2;

erreur(numerr)

end

end; (*TSTROTCMP*)

procedure trtschlnd;

var crf:packed array [1..nnmmx1] of char;

crfcrx:0..nnmmx1;

begin

for crfcrx:=1 to nnmmx1 do crf[crfcrx]:=' ';

getsgncar(stk[stkcrx]);

crfcrx:=0;

while (crtcar <> ')') and (crfcrx < nnmmx1)

do begin

crfcrx:=crfcrx+1;

crf[crfcrx]:=crtcar;

getsgncar(stk[stkcrx])

end;

if crtcar <> ')') then begin

numerr:=1;

erreur(numerr)

end;

rtbcrx:=1;

while (rtb[rtbcrx].rnm <> crf) and (rtbcrx <= rtbmx1)

do rtbcrx:=rtbcrx+1;

if rtbcrx > rtbmx1 then begin

numerr:=9;

erreur(numerr)

end

else begin

ntbkpx:=rtb[rtbcrx].ntbptr;

lnd:=ntb[ntbkpx].rtp;

crtlev:=ntb[lnd].lnb+1

end

end; (*TRTSCHLND*)

procedure trtrot;

begin

tstrotcmp;

 rtb[rtbkpx].ftbptr:=rtbkpx;

 rtb[rtbkpx].ntbptr:=ntbfrp;

for crncrx:=1 to nmmx1

do rtb[rtbkpx].nnm[crncrx]:=crn[crncrx];

 ntbcx:=ntbfrp;

 ntb[ntbcx].lnb:=crtlev;

 ntb[ntbcx].fap:=rtbkpx;

 ntb[ntbcx].rtp:=ntbcx;

 ntb[ntbcx].rcp:=0;

for crncrx:=1 to nmmx1

do ntb[ntbcx].nnm[crncrx]:=crn[crncrx];

 lnd:=ntbcx;

 ntbfrp:=ntb[ntbcx].nxp;

 ntb[ntbcx].nxp:=0;

 crtlev:=crtlev+1

end; (*TRTROT*)

procedure trtnod;

var oldlev:integer;

 ntbidx:0..ntbm1;

procedure tstnodcmp;

var cond:boolean;

begin

 ntbidx:=lnd;

 oldlev:=crtlev;

while ntb[ntbidx].lnb<>crtlev-1

do ntbidx:=ntb[ntbidx].fap;

 oldlev:=ntb[ntbidx].lnb;

 ntbidx:=ntb[ntbidx].nxp;

 cond:=true;

while cond and (ntbidx<>0)

do

if ntb[ntbidx].lnb>oldlev

then

if (ntb[ntbidx].lnb=crtlev) and (ntb[ntbidx].nnm=crn)

then begin

 numerr:=4;

 erreur(numerr)

end

else ntbidx:=ntb[ntbidx].nxp

else cond:=false;

end; (*TSTNODCMP*)

begin

tstnodcmp;

 ntbcx:=ntbfrp;

for crncrx:=1 to nmmx1

do ntb[ntbcx].nnm[crncrx]:=crn[crncrx];

 ntb[ntbcx].lnb:=crtlev;

 ntb[ntbcx].rcp:=0;

 ntb[lnd].nxp:=ntbcx;

 ntbidx:=lnd;


```

oldlev:=crtlev;
while ntb[ntbidx].lnb > oldlev
do ntbidx:=ntb[ntbidx].fap;

if ntb[ntbidx].lnb=oldlev
then ntb[ntbcrx].fap:=ntb[ntbidx].fap;
if ntb[ntbidx].lnb=oldlev-1
then ntb[ntbcrx].fap:=ntbidx;

ntb[ntbcrx].rtp:=ntb[ntbidx].rtp;
lnd:=ntbfrp;
ntb[ntb[ntbcrx].rtp].rtp:=lnd;
ntbfrp:=ntb[ntbcrx].nxp;
ntb[lnd].nxp:=0;
crtlev:=crtlev+1
end; (*TRTNOD*)

```

```

procedure trtnnm;

```

```

begin
  setcrn;
  if crtlev = 0 then trttrot
  else trtnod
end; (*TRTNM*)

```

```

procedure trtcre;

```

```

begin
  rnb:=0;
  if crtcar='V' then begin
    vrfsw:=true;
    getsgncar(stk[stkcrx])
  end

  else if crtcar='R' then begin
    vrfsw:=false;
    getsgncar(stk[stkcrx])
  end

  else begin
    numerr:=1;
    erreur(numerr)
  end;

  if crtcar=',' then getsgncar(stk[stkcrx])
  else begin
    numerr:=1;
    erreur(numerr)
  end;

  if crtcar='(' then begin
    trtschlnd;
    getsgncar(stk[stkcrx])
  end;

  trtnnm;
  if crtcar<>','
  then
    begin
      odtcrx:=0;
      if vrfsw=true then trtconval(lnd)
      else trtconnod(lnd)
    end
end

```


end; (*TRTCRE*)

procedure trtend;

begin

if crtcar=';' then begin

then begin

crtlev:=crtlev-1;

if crtlev<0

then begin

numerr:=7;

erreur(numerr)

end

end

else begin

for crncrx:=1 to nmmx1

do crn[crncrx]:=' ';

crncrx:=0;

while (crtcar<>'') and (crncrx<>nmmx1)

do begin

crncrx:=crncrx+1;

crn[crncrx]:=crtcar;

getsgncar(stk[stkcrx])

end;

if crtcar<>' ' then

then

begin

numerr:=1;

erreur(numerr)

end;

ntbcrx:=1nd;

while (ntb[ntbcrx].nnm<>crn) and (ntb[ntbcrx].lnb<>0)

do ntbcrx:=ntb[ntbcrx].fap;

if (ntb[ntbcrx].lnb=0) and (ntb[ntbcrx].nnm=crn)

then crtlev:=0;

if (ntb[ntbcrx].lnb<>0) and (ntb[ntbcrx].nnm=crn)

then crtlev:=ntb[ntbcrx].lnb;

if (ntb[ntbcrx].lnb=0) and (ntb[ntbcrx].nnm<>crn)

then begin

numerr:=11;

erreur(numerr)

end

end

end; (*TRTEND*)

procedure trtins;

var sob:char;

fstidx,1stidx,sobfpl:0..ntbmx1;

fstlev:integer;

procedure trtyb;

var i:integer;

begin

askcrx:=askcrx-1;

fstidx:=ask[askcrx];

sobfpl:=ntbfrp;

fstlev:=ntb[fstidx].lnb;

ntbcx:=ntb[fstidx].nxp;

if ntb[ntbcx].lnb<=fstlev

then lstidx:=fstidx;

while ntb[ntbcx].lnb > fstlev

do begin

lstidx:=ntbcx;

ntbcx:=ntb[ntbcx].nxp

end;

for i:=1 to nmmx1

do ntb[sobfpl].nnm[i]:=crn[i];

ntb[sobfpl].lnb:=fstlev;

ntb[sobfpl].fap:=ntb[fstidx].fap;

ntb[sobfpl].rtp:=ntb[fstidx].rtp;

ntb[sobfpl].rcp:=0;

ntbfrp:=ntb[sobfpl].nxp;

ntb[sobfpl].nxp:=ntb[lstidx].nxp;

ntb[lstidx].nxp:=sobfpl

end; (*TRTYB*)

procedure trtyo;

var i:integer;

begin

askcrx:=askcrx-1;

fstidx:=ask[askcrx];

sobfpl:=ntbfrp;

fstlev:=ntb[fstidx].lnb;

ntbcx:=ntb[fstidx].nxp;

if ntb[ntbcx].lnb<=fstlev

then lstidx:=fstidx;

while ntb[ntbcx].lnb>fstlev

do begin

lstidx:=ntbcx;

ntbcx:=ntb[ntbcx].nxp

end;

for i:=1 to nmmx1

do ntb[sobfpl].nnm[i]:=crn[i];

ntb[sobfpl].lnb:=fstlev+1;

ntb[sobfpl].fap:=fstidx;

ntb[sobfpl].rcp:=0;

if fstlev = 0

then ntb[sobfpl].rtp:=fstidx

else ntb[sobfpl].rtp:=ntb[fstidx].rtp;

ntbfrp:=ntb[sobfpl].nxp;

ntb[sobfpl].nxp:=ntb[lstidx].nxp;

ntb[lstidx].nxp:=sobfpl

end; (*TRTYO*)

procedure trtel;

var i:integer;

begin

askcrx:=askcrx-1;

fstidx:=ask[askcrx];

sobfpl:=ntbfrp;

fstlev:=ntb[fstidx].lnb;

for i:=1 to nmmx1

do ntb[sobfpl].nnm[i]:=crn[i];

ntb[sobfpl].lnb:=fstlev+1;

ntb[sobfpl].fap:=fstidx;

ntb[sobfpl].rcp:=0;

if fstlev=0 then ntb[sobfpl].rtp:=fstidx

else ntb[sobfpl].rtp:=ntb[fstidx].rtp;

ntbfrp:=ntb[sobfpl].nxp;

ntb[sobfpl].nxp:=ntb[fstidx].nxp;

ntb[fstidx].nxp:=sobfpl

end; (*TRTEL*)

begin

rn timer:=0;

if crtcdr = 'V' then begin

vrfs:=true;

getsgncar(stk[stkcrx]);

end

else if crtcdr = 'R'

then begin

vrfs:=false;

getsgncar(stk[stkcrx]);

end

else begin

numerr:=1;

erreur(numerr)

end;

if crtcdr = '('

then begin

getsgncar(stk[stkcrx]);

trnid;

if (moy=false) and (nnb>1)

then begin

numerr:=6;

erreur(numerr)

end

else if nnb=0

then begin

numerr:=5;

erreur(numerr)

end;

getsgncar(stk[stkcrx]);

end

else begin

numerr:=1;

erreur(numerr)

end;

if crtcdr = 'Y'

then begin

getsgncar(stk[stkcrx]);

case crtcdr of 'B':sob:='B';

'O':sob:='O';

end;

if (crtcdr<>'O') and (crtcdr<>'B')

then begin

```

        numerr:=1;
        erreur(numerr)
    end
end
else if crtcar = 'E'
    then begin
        getsgncar(stk[stkcrx]);
        if crtcar = 'L' then sob:='L'
        else begin
            numerr:=1;
            erreur(numerr)
        end
    end
    else begin
        numerr:=1;
        erreur(numerr)
    end;
getsgncar(stk[stkcrx]);
if crtcar=', '
    then begin
        getsgncar(stk[stkcrx]);
        setcrn
    end
    else begin
        numerr:=1;
        erreur(numerr)
    end;
repeat
    case sob of 'B':trtyb;
                'O':trtyo;
                'L':trtel
    end;
    if crtcar<>' '
    then
        begin
            odtkrx:=0;
            if vrfsw=false
            then trtconnod(sobfpl)
            else trtconval(sobfpl)
        end
    until askcrx=askkpx
end; (*TRTINS*)

```

procedure trtmod;

```

var fstidx:0..ntbmx1;
    prsnm,prscon:boolean;

```

procedure modnrm;

```

begin
    for nmcrx:=1 to nmmx1
        do ntb[fstidx].nm[nmcrx]:=crn[nmcrx]
    end; (*MODNNM*)

```


procedure modcon;

```
begin
  if ntb[fstidx].rcp=0
  then
    begin
      odtcrx:=0;
      idtcrx:=stk[1].crp;
      crtcar:=stk[1].buf.idt[idtcrx];
      trtconnod(fstidx)
    end
  else
    begin
      rtbcrx:=ntb[ntb[fstidx].rtp].fap;
      ftbcrx:=rtb[rtbcrx].ftbptr;
      for nmmcrx:=1 to nmmx1
      do stk[2].fnn[nmmcrx]:=ntb[ntb[fstidx].rtp].nmm[nmmcrx];
      stk[2].buf.nxr:=ntb[fstidx].rcp;
      repeat
        ftb[ftbcrx][stk[2].buf.nxr]:=true;
        rdlin(stk[2])
      until stk[2].buf.nxr=0;
      rnb:=0;
      idtcrx:=stk[1].crp;
      crtcar:=stk[1].buf.idt[idtcrx];
      odtcrx:=0;
      trtconnod(fstidx)
    end
  end; (*MODCON*)
```

```
begin
  if crtcar='('
  then begin
    getsnrcar(stk[stkcrx]);
    trtnid;
    getsnrcar(stk[stkcrx])
  end
  else begin
    numerr:=1;
    erreur(numerr)
  end;
  if (mny=false) and (nnb>1)
  then begin
    numerr:=6;
    erreur(numerr)
  end
  else if nnb=0
  then begin
    numerr:=5;
    erreur(numerr)
  end;
  if crtcar='' then prsnnm:=false
  else begin
    prsnnm:=true;
    setcrn
  end;
  if crtcar=';' then prscon:=false
  else prscon:=true;
  stk[1].crp:=idtcrx;
  repeat
    askcrx:=askcrx-1;
    fstidx:=ask[askcrx];
    if prsnnm=true then modnrm;
```

```

    if prscon=true then modcon
    until askcrx=askkpx
end; (*TRTMOD*)

```

```

procedure addcon;

```

```

var fstidx:0..ntbmx1;
    fstlev:integer;

```

```

begin
    stk[1].crp:=idtcx;
    repeat
        askcrx:=askcrx-1;
        fstidx:=ask[askcrx];
        fstlev:=ntb[fstidx].lnb;
        if ntb[fstidx].rcp=0
        then
            begin
                rnb:=0;
                idtcx:=stk[1].crp;
                crtcx:=stk[stkcrx].buf.idt[idtcx];
                for odtcx:=1 to odtmx1
                    do outrec.odt[odtcx]:= ' ';
                odtcx:=0;
                trtconnod(fstidx)
            end
        else
            begin
                if fstlev=0
                then for nncrx:=1 to nncmx1
                    do stk[2].fnm[nncrx]:=ntb[fstidx].nrm[nncrx]
                else for nncrx:=1 to nncmx1
                    do stk[2].fnm[nncrx]:=ntb[ntb[fstidx].rtp].nrm[nncrx];
                stk[2].buf.nxr:=ntb[fstidx].rcp;
                rnb:=0;
                repeat
                    crtnxr:=stk[2].buf.nxr;
                    rdlin(stk[2]);
                    rnb:=rnb+1
                until stk[2].buf.nxr=0;
                odtcx:=1;
                while stk[2].buf.idt[odtcx] <> chr(3)
                do
                    begin
                        outrec.odt[odtcx]:=stk[2].buf.idt[odtcx];
                        odtcx:=odtcx+1
                    end;
                odtcx:=odtcx-1;
                idtcx:=stk[1].crp;
                crtcx:=stk[stkcrx].buf.idt[idtcx];
                trtconnod(fstidx)
            end
        until askcrx=askkpx
    end; (*ADDCON*)

```



```

procedure trtadd;
begin
  if crtcar='('
  then begin
    getsgncar(stk[stkcrx]);
    trtnid;
    getsgncar(stk[stkcrx])
  end
  else begin
    numerr:=1;
    erreur(numerr)
  end;
  if (mny=false) and (nnb>1)
  then begin
    numerr:=6;
    erreur(numerr)
  end
  else if nnb=0
  then begin
    numerr:=5;
    erreur(numerr)
  end
  else addcon
end; (*TRTADD*)

```

```

procedure trtdel;

```

```

procedure del;

```

```

var fstidx, lstidx: 0..ntbmx1;
    fstlev: integer;
    idx : integer;

```

```

procedure delall;

```

```

begin
  rtbcrx:=ntb[fstidx].fap;
  ftbcrx:=rtb[rtbcrx].ftbptr;
  for idx:=1 to nmmx1
  do rtb[rtbcrx].nnm[idx]:= ' ';
  for fplcrx:=nnb+1 to fplmx1
  do ftb[ftbcrx][fplcrx]:=true;
  ntbcx:=fstidx;
  repeat
    for nnmcx:=1 to nmmx1
    do ntb[ntbcrx].nnm[nnmcx]:= ' ';
    ntbcx:=ntb[ntbcrx].nxp
  until ntbcx=0;
  lnd:=ntb[fstidx].rtp;
  ntb[lnd].nxp:=ntbfrp;
  ntbfrp:=fstidx
end; (*DELALL*)

```

```

procedure delnds;
var fini : boolean;

begin
  rtbcrx:=ntb[ntb[fstidx].rtp].fap;
  ftbcrx:=rtb[rtbcrx].ftbptr;
  for nnmcx:=1 to nnmx1
    do stk[1].fnn[nnmcx]:=ntb[ntb[fstidx].rtp].nnm[nnmcx];
  ntbcx:=fstidx;

  fini:=false;
  repeat
    stk[1].buf.nxr:=ntb[ntbcrx].rcp;
    for nnmcx:=1 to nnmx1
      do ntb[ntbcrx].nnm[nnmcx]:=' ';
    if stk[1].buf.nxr<>0
      then repeat
        ftb[ftbcrx][stk[1].buf.nxr]:=true;
        rdlin(stk[1])
        until stk[1].buf.nxr=0;
    lstidx:=ntbcrx;

    if ntb[ntbcrx].nxp=0
      then fini:=true
      else ntbcx:=ntb[ntbcrx].nxp;

    if not fini and (ntb[ntbcrx].lnb<=fstlev)
      then fini:=true
  until fini;

  ntbcx:=ntb[fstidx].fap;
  repeat
    ntbkpx:=ntbcrx;
    ntbcx:=ntb[ntbcrx].nxp
  until ntbcx=fstidx;
  ntb[ntbkpx].nxp:=ntb[lstidx].nxp;

  ntb[lstidx].nxp:=ntbfrp;
  ntbfrp:=fstidx;
end; (*DELND*)

```

```

begin
  repeat
    askcrx:=askcrx-1;
    fstidx:=ask[askcrx];
    fstlev:=ntb[fstidx].lnb;
    if fstlev=0
      then delall
      else delnds
  until askcrx=askkpx;
  for nnmcx:=1 to nnmx1
    do stk[1].fnn[nnmcx]:=fsk[1][nnmcx];
end; (*DEL*)

```

```

begin (*TRTDEL*)
  if crtdcr='('
    then
      begin
        getsqncar(stk[stkcrx]);

```



```

        trtnid
    end
else
    begin
        numerr:=1;
        erreur(numerr)
    end;
if (mny=false) and (nnb>1)
then
    begin
        numerr:=6;
        erreur(numerr)
    end
else if nnb=0
    then
        begin
            numerr:=5;
            erreur(numerr)
        end
    else del
end; (*TRTDEL*)

```

(*S+*)

unit decl;

interface

```
const askmx1 = 10;
      nnmmx1 = 6;
      fsmx1 = 5;
      ftbmx1 = 4;
      fplmx1 = 1535;
      ntbmx1 = 160;
      recmx1 = 82;
      odtmx1 = 80;
      rtbmx1 = 4;
      stkmx1 = 10;
      idtmx1 = 80;
      sskmx1 = 10;
      bnb = 3;
      nse = 5;
      dsknmemx1 = 10;

      blk1gh = 512;
      reclgh = 82;
      rnbblk = 6;
      blnmx1 = 640;
```

```
type fskent = packed array [1..nnmmx1] of char;
      fpl = packed array [0..fplmx1] of boolean;
      lincom = packed array [1..idtmx1] of char;
      ntcent = record
          nrm:packed array [1..nnmmx1] of char;
          lnb:integer;
          rtp:0..ntbmx1;
          nxp:0..ntbmx1;
          fap:0..ntbmx1;
          rcp:integer
      end; (*END RECORD*)
      rtcent = record
          rnm :packed array [1..nnmmx1] of char;
          ntbp:0..ntbmx1;
          ftbp:0..ftbmx1
      end; (*END RECORD*)

      rec = record
          odt:packed array [1..odtmx1] of char;
          nxr:integer
      end; (*END RECORD*)

      inrec = record
          idt:packed array [1..idtmx1] of char;
          nxr:integer
      end; (*END RECORD*)

      stkent = record
          frm:packed array [1..nnmmx1] of char;
          crp:0..idtmx1;
          buf:inrec
      end; (*END RECORD*)
```



```

askent = record
    ntbcx:0..ntbmx1;
    askkpx:0..askmx1
end;

dskrec = record
    dskname : packed array [1..10] of char;
    blk1stptr : integer
end;

```

```

var askcrx : integer;
askkpx : integer;
ask : array[1..askmx1] of integer;
crn : packed array [1..nnmmx1] of char;
fsk : array [1..fskmx1] of fskent;
ftb : array [1..ftbmx1] of fpl;
ntb : array [1..ntbmx1] of ntcent;
rtb : array [1..rtbmx1] of rtcent;
stk : array [1..stkmx1] of stkent;
crncrx : integer;
fskcrx : integer;
ftbcx : integer;
ftbkpx : integer;
fplcrx : integer;
fplkpx : integer;
ntbcx : integer;
ntbkpx : integer;
nnmcx : integer;
odtcx : integer;
rtbcx : integer;
rtbkpx : integer;
rnmcrx : integer;
stkcrcx : integer;
fnmcrx : integer;
idtcx : integer;
trsarg,actarg : array [0..4,1..5] of integer;
trsnid,actnid : array [0..5,1..6] of integer;
crtlev : integer;
crtcar : char;
crtncr : integer;
lnd : 0..ntbmx1;
mny : boolean;
nnb : integer;
ntbfrp : 0..ntbmx1;
rnb : integer;
vrfs : boolean;
fo : file of char;
lgnnbr : integer;
numerr : integer;
outrec : rec;
str : stkent;
flenme : string[15];
ask:array[1..askmx1] of askent;
askcrx:0..askmx1;
fstnbr : integer;
secnbr : integer;

i : integer;
car : char;
rcslav : integer;

preblk : integer;

```

```

profile : integer;
prcop   : boolean;
flenum  : integer;

dskbuf  : packed array [1..1] of dskrec;
ptrlst  : packed array [1..256] of integer;
flebuf  : packed array [1..rnbbk] of inrec;
fstblk  : packed array [1..1] of integer;
password : packed array [1..10] of char;

```

```

procedure erreur(var numero:integer);
procedure readf0(var tampon:lincom);
procedure verifdsk;
procedure chgblktbl(var flenum:integer);
procedure rchblktbl;
procedure fndblk(var blkadr:integer);
procedure free(var flenme:fskent;
               var enrnum:integer);
procedure readfile(flenum,enrnum:integer;
                  var tampon:inrec);
procedure writefile(flenum,enrnum:integer;
                   var tampon:rec);
procedure rdlin(var str:stkent);
procedure getcar(var str:stkent);
procedure getsgncar(var str:stkent);
procedure schfpl(var crtadr:integer);
procedure wrtrec(var crtadr:integer);
procedure putcar(var crtadr:integer);
procedure trtnid;

```

implementation

```
procedure erreur;
```

```

begin
  case numero of
    1:writeln('LIGNE ',lgnnbr,' ERREUR DE SYNTAXE');
    2:writeln('LIGNE ',lgnnbr,' FICHER DEJA CREE');
    3:writeln('LIGNE ',lgnnbr,' TABLE DES RACINES PLEINE');
    4:writeln('LIGNE ',lgnnbr,' NOEUD DEJA EXISTANT');
    5:writeln('LIGNE ',lgnnbr,' AUCUN NOEUD IDENTIFIE');
    6:writeln('LIGNE ',lgnnbr,' FAMILLE DE NOEUDS IDENTIFIEE');
    7:writeln('LIGNE ',lgnnbr,' FIN DE CREATION ERRONEE');
    8:writeln('LIGNE ',lgnnbr,' DEBORDEMENT TABLE FPL');
    9:writeln('LIGNE ',lgnnbr,' DEBORDEMENT TABLE NTB');
    10:writeln('PLUS DE BLOCS LIBRES!!!');
    11:writeln('LIGNE ',lgnnbr,' NOM DE NOEUD INEXISTANT');
  end; (*CASE*)
  exit(program)
end; (*BEGIN*)

```

```
procedure readf0;
```

```
var j:integer;
```

```

begin
  lgnnbr:=lgnnbr+1;
  writeln;
  writeln('LIGNE DE COMMANDE NUMERO ',lgnnbr);
  writeln;

```



```

for idtcx:=1 to idtmx1
  do tampon[idtcx]:= ' ' ;

idtcx:=0;
repeat
  idtcx:=idtcx+1;
  tampon[idtcx]:=f0^;
  if (idtcx<>1) or not eoln(f0)
    then get(f0)
until (idtcx=idtmx1) or eoln(f0);
if (idtcx<idtmx1) then
  for j:=idtcx+1 to idtmx1
    do tampon[j]:= ' '
end;

```

```

procedure verifdsk;      ( Cette procédure vérifie la présence de
                          la disquette dont l'utilisateur a donné
                          le nom. )
var caractere:char;

```

```

begin
  repeat
    unitread(5,dskbuf,12,0);
    writeln('VERIFIEZ SI LA DISQUETTE ');
    for i:=1 to 10
      do write(password[i]);
    writeln;
    writeln('SE TROUVE DANS LE DRIVE 2. ');
    writeln('TAPEZ <RETURN> POUR CONTINUER. ');
    readln(caractere)
  until dskbuf[11].dskname = password;
end; (*VERIFDSK*)

```

```

procedure chgblktbl;      ( Cette procédure charge en mémoire la table
                          des blocs d'un fichier )

```

```

var caractere : char;

```

```

begin
  if prcfile <> 0
    then unitwrite(5,ptrlst,512,prcfile);
  prcfile:=flenum;

  unitread(5,ptrlst,512,flenum)

end; (*CHGBLKTBL*)

```

```

procedure rchblktbl;      ( Cette procédure recharge sur disque le bloc
                          se trouvant dans le buffer et la table des
                          blocs du fichier correspondant )
begin
  unitwrite(5,flebuf,492,prcblk);
  unitwrite(5,ptrlst,512,prcfile)
end; (*RCHBLKTBL*)

```

```

procedure fndblk;
                                ( Cette procédure met dans
                                blkadr l'adresse du premier bloc libre et
                                met à jour cette adresse. )
begin
  blkadr:=dskbuf[11].blk1stptr;
  if blkadr <> 280
  then
    begin
      unitread(5,fstblk,2,blkadr);
      dskbuf[11].blk1stptr:=fstblk[11];
      unitwrite(5,dskbuf,12,0)
    end
  end; (*FNDBLK*)

```

```

procedure free;

```

```

begin
  rtbcrx:=0;
  repeat
    rtbcrx:=rtbcrx+1;
    if rtb[rtbcrx].rnm=flenme
    then ftbcrx:=rtb[rtbcrx].ftbptr
  until (rtbcrx=rtbmx1) or (rtb[rtbcrx].rnm=flenme);
  ftb[ftbcrx].enrnum:=true;
end; (*FREE*)

```

```

procedure readfile;

```

```

var idx,
    blknum,
    blkadr:integer;
    caractere:char;

```

```

begin
  if flenum <> prcfile
  then chgblk1(flenum);

  blknum:=enrnum div rnbblk + 1;
  blkadr:=ptr1st[blknum];
  if blkadr <> prcblk
  then
    begin
      if prcop
      then
        begin
          prcop:=false;
          unitwrite(5, flebuf, 492, prcblk)
        end;
      prcblk:=blkadr;
      unitread(5, flebuf, 492, blkadr)
    end;

  enrnum:=enrnum mod rnbblk+1;
  for idx:=1 to idtmx1
  do tampon.idt[idx]:=flebuf[enrnum].idt[idx];
  tampon.nxr:=flebuf[enrnum].nxr;
end; (*READFILE*)

```


procedure writefile;

var blkadr,
 blknum,
 idx : integer;
 caractere:char;

begin

if flenum <> prcfla
then chgblktbl(flenum);

blknum:=enrnum div rnbblk + 1;

blkadr:=ptrlst[blknum];

if blkadr = 0

then fndblk(blkadr);

if blkadr = 280

then

begin

numerr:=10;

erreur(numerr)

end

else

begin

ptrlst[blknum]:=blkadr;

if blkadr <> prcblk

then

begin

if prcop = true

then unitwrite(5, flebuf, 492, prcblk);

prcblk:=blkadr;

unitread(5, flebuf, 492, blkadr)

end;

enrnum:=enrnum mod rnbblk + 1;

for idx:=1 to idtmx1

do flebuf[enrnum].idt[idx]:=tampon.odt[idx];

flebuf[enrnum].nxx:=tampon.nxx;

prcop:=true

end

end; (*WRITEFILE*)

procedure rdlin;

var flenum : integer;

begin

for idtcx := 1 to idtmx1 do str.buf.idt[idtcx]:= ' ';

if str.fnm = fsk[1]

then

readf0(str.buf.idt)

else

begin

if str.fnm = fsk[2] then flenum:=1

else if str.fnm = fsk[3] then flenum:=2

else if str.fnm = fsk[4] then flenum:=3

else if str.fnm = fsk[5] then flenum:=4;

readfile(flenum, str.buf.nxx, str.buf)

end;

idtcx:=0

```
end; (*RDLIN*)
```

```
procedure getcar;
```

```
begin
  if idtcx = idtmx1
  then
    begin
      if str.fnm=fsk[1] then get(f0);
      rdlin(str)
    end;
    idtcx:=idtcx+1;
    crtcar:=str.buf.idt[idtcx]
end; (*GETCAR*)
```

```
procedure getsncar;
```

```
begin
  repeat
    getcar(str)
  until crtcar <> ' '
end; (*GETSGNCAR*)
```

```
procedure schfpl;
```

```
begin
  ntbkpx:=ntb[crtadr].rtp;
  rtbkpx:=ntb[ntbkpx].fap;
  ftbkpx:=rtb[rtbkpx].ftbptr;
  fplcrx:=bnb + 1;
  while (ftb[ftbkpx][fplcrx]=false) and (fplcrx<=fplmx1)
  do fplcrx:=fplcrx+1;
  if fplcrx > fplmx1 then begin
    numerr:=8;
    erreur(numerr)
  end; (*IF*)

  fplkpx:=fplcrx;
  ftb[ftbkpx][fplcrx]:=false
end; (*SCHFPL*)
```

```
procedure wrtrec;
```

```
var crtfnm:fskent;
    i:integer;
    flenum:integer;
```

```
begin
  schfpl(crtadr);
  rnb:=rnb+1;
  if rnb=1 then begin
    ntbcrtadr].rcp:=fplkpx;
    crtncr:=fplkpx;
    schfpl(crtadr)
  end; (*IF*)
  outrec.ncr:=fplkpx;

  if ntbcrtadr].lnb<>0
  then for i:=1 to nmmx1
    do crtfnm[i]:=ntb[ntbcrtadr].rtp].nmm[i]
```



```

else for i:=1 to nmmx1
  do crtfnm[i]:=ntb[crtadr].nm[i];

  if crtfnm = fsk[2] then flenum:=1
else if crtfnm = fsk[3] then flenum:=2
else if crtfnm = fsk[4] then flenum:=3
else if crtfnm = fsk[5] then flenum:=4;
writefile(flenum,crtmxr,outrec);
crtmxr:=outrec.nxr
end; (*WRTREC*)

```

procedure putcar;

```

begin
  if odcrx = odtmx1
  then begin
    wtrrec(crtadr);
    for odcrx:=1 to odtmx1
      do outrec.odt[odcrx]:= ' ';
    odcrx:=0
  end; (*IF*)
  odcrx:=odcrx+1;
  outrec.odt[odcrx]:=crtcar;

end; (*PUTCAR*)

```

procedure trtnid;

```

const nskmx1=10;
      cmnmx1=10;

```

type name = packed array [1..nmmx1] of char;

```

var ste,oldste:integer;
    nsk:array[1..cmnmx1] of name;
    nskcrx : 0..nskmx1;
    nme:name;
    nmecrx:0..nmmx1;
    lsk:array [1..cmnmx1] of integer;
    cmncrx:0..cmnmx1;
    cmnkpx:0..cmnmx1;
    levnbr:integer;
    code: integer;

```

function codnid:integer;

```

begin
  if (crtcar <> '!') and (crtcar <> '?') and (crtcar <> '.')
    and (crtcar <> ')')
    and (crtcar <> '0')
    and (crtcar <> '1')
    and (crtcar <> '2')
    and (crtcar <> '3')
    and (crtcar <> '4')
    and (crtcar <> '5')

```

```

and (crtcar <> '6')
and (crtcar <> '7')
and (crtcar <> '8')
and (crtcar <> '9')
and (crtcar <> 'A')
and (crtcar <> 'B')
and (crtcar <> 'C')
and (crtcar <> 'D')
and (crtcar <> 'E')
and (crtcar <> 'F')
and (crtcar <> 'G')
and (crtcar <> 'H')
and (crtcar <> 'I')
and (crtcar <> 'J')
and (crtcar <> 'K')
and (crtcar <> 'L')
and (crtcar <> 'M')
and (crtcar <> 'N')
and (crtcar <> 'O')
and (crtcar <> 'P')
and (crtcar <> 'Q')
and (crtcar <> 'R')
and (crtcar <> 'S')
and (crtcar <> 'T')
and (crtcar <> 'U')
and (crtcar <> 'V')
and (crtcar <> 'W')
and (crtcar <> 'X')
and (crtcar <> 'Y')
and (crtcar <> 'Z')

```

```

then codnid:=6

```

```

else case crtcar of '!' :codnid:=1;

```

```

'*' :codnid:=2;

```

```

'.' :codnid:=3;

```

```

')' :codnid:=5;

```

```

'1','2','3','4','5','6','7','8','9','0':codnid:=4;

```

```

'A','B','C','D','E','F','G','H','I','J':codnid:=4;

```

```

'K','L','M','N','O','P','Q','R','S','T':codnid:=4;

```

```

'U','V','W','X','Y','Z':codnid:=4

```

```

end

```

```

end; (*CODNID*)

```

```

procedure trtone;

```

```

begin

```

```

  mny:=false

```

```

end; (*TRTONE*)

```

```

procedure levdst;

```

```

begin

```

```

  levnbr:=levnbr+1

```

```

end; (*LEV DST*)

```

```

procedure trsnnmlev;

```

```

begin

```

```

  cmncrx:=cmncrx+1;

```

```

  levnbr:=levnbr+1

```

```

end; (*TRSNM LEV*)

```


procedure inttrtnnm;

begin

nmecrx:=1;

lsk[cmncrx]:=levnbr;

nsk[cmncrx][nmecrx]:=crtcar

end; (*INTTRTNM*)

procedure trtsta;

begin

lsk[cmncrx]:=-1;

nsk[cmncrx][nmecrx]:=crtcar

end; (*TRTSTA*)

procedure itetrttnnm;

begin

nmecrx:=nmecrx+1;

nsk[cmncrx][nmecrx]:=crtcar

end; (*ITETRTNNM*)

procedure updask;

begin

nbn:=nbn+1;

ask[askcrx]:=ntbkpx;

if nbn = 1 then askkpx:=askcrx;

askcrx:=askcrx+1

end; (*UPDASK*)

procedure anaway;

var idnnnm:boolean;

i,aux:integer;

begin

if cmncrx = 1

then

begin

i:=0;

repeat

ntbcx:=ntb[ntbcx].fap;

i:=i+1

until ntb[ntbcx].lnb=0;

if i = lsk[cmncrx]

then updask

end;

idnnnm:=true;

while (cmncrx>1) and idnnnm

do

begin

i:=0;

if lsk[cmncrx-1]=-1

then aux:=lsk[cmncrx]

else aux:=lsk[cmncrx]-lsk[cmncrx-1];

repeat

i:=i+1;

ntbcx:=ntb[ntbcx].fap

until (i=aux) or (ntb[ntbcx].lnb=0);

cmncrx:=cmncrx-1;

```

    if (ntb[ntbcrx].nnm <> nsk[cmncrx]) or (i<aux)
        then idnnm:=false;
    if idnnm and (cmncrx=1)
        then if (lsk[cmncrx]=-1) or (lsk[cmncrx]=ntb[ntbcrx].lnb)
            then updask;
        end;
    ntbcrx:=ntbkpx+1;
    cmncrx:=cmnkpx;
end; (*ANAWAY*)

```

procedure schnds;

var trouve:boolean;

```

begin
    ntbcrx:=1;
    cmnkpx:=cmncrx;
    trouve:=false;

    while ntbcrx<=ntbmxl
        do
            begin
                if ntb[ntbcrx].nnm <> nsk[cmnkpx]
                    then ntbcrx:=ntbcrx+1
                else
                    begin
                        trouve:=true;
                        ntbkpx:=ntbcrx;
                        if (cmncrx = 1) and (lsk[1] = -1)
                            then
                                begin
                                    updask;
                                    ntbcrx:=ntbcrx+1;
                                end
                            else anaway;
                    end
                end;
            if (ntbcrx > ntbmx1) and not trouve
                then
                    begin
                        numerr:=11;
                        erreur(numerr);
                    end
            end;
end; (*SCHNDS*)

```

```

begin
    levnbr:=0;
    nnb:=0;
    mny:=true;
    ste:=0;
    for cmncrx:=1 to cmnmxl
        do
            for nmecrx:=1 to nmnmxl
                do nsk[cmncrx][nmecrx]:=' ';
            cmncrx:=1;
            nmecrx:=1;
            while ste >= 0
                do
                    begin
                        oldste:=ste;
                        code:=codnid;
                        ste:=trsnid[oldste,code];
                        case actnid[oldste,code] of -1:begin
                                numerr:=1;
                                erreur(numerr)

```



```
end;  
1:trtone;  
3:levdst;  
43:trsnmlev;  
4:inttrtnnm;  
24:trtsta;  
44:itetrttnnm;  
45:schnds
```

```
end;  
  getsnrcar(stk[stkcrx])  
end  
end; (*TRTNID*)
```

```
begin  
end.
```

(*S+*)

unit inter;

interface

(*SU APPLE2:DECLIB.CODE *)

uses decl;

procedure chgntb(var crtfrm:fskent);

procedure trtint;

procedure trtter;

implementation

procedure chgntb;

var i,j,k:integer;

fstidx,lstidx:0..ntbmx1;

nxrkpx:integer;

procedure setntbent;

var conversion:packed array[1..21] of char;

begin

nnmcx:=1;

for idtcx:=1 to j

do

begin

ntb[ntbcx].nnm[nnmcx]:=stk[stkcx].buf.idt[idtcx];

nnmcx:=nnmcx+1

end;

i:=j+1;

j:=j+2;

for idtcx:=i to j

do conversion[idtcx-j+2]:=stk[stkcx].buf.idt[idtcx];

moveleft(conversion,ntb[ntbcx].lmb,2);

i:=j+1;

j:=j+2;

for idtcx:=i to j

do conversion[idtcx-j+2]:=stk[stkcx].buf.idt[idtcx];

moveleft(conversion,ntb[ntbcx].fap,2);

i:=j+1;


```

j:=j+2;
for idtcrx:=i to j
  do conversion[idtcrx-j+2]:=stk[stkcrx].buf.idt[idtcrx];
moveleft(conversion,ntb[ntbcrx].npx,2);

i:=j+1;
j:=j+2;
for idtcrx:=i to j
  do conversion[idtcrx-j+2]:=stk[stkcrx].buf.idt[idtcrx];
moveleft(conversion,ntb[ntbcrx].rtp,2);

i:=j+1;
j:=j+2;
for idtcrx:=i to j
  do conversion[idtcrx-j+2]:=stk[stkcrx].buf.idt[idtcrx];
moveleft(conversion,ntb[ntbcrx].rcp,2);

```

end; (*SETNTBENT*)

begin

```

k:=1;
i:=1;
j:=nnmmx1;

```

```

for nnmcx:=1 to nnmmx1
  do stk[stkcrx].fnn[nnmcx]:=crtfnn[nnmcx];
stk[stkcrx].buf.nxr:=bnb;

```

rdlin(stk[stkcrx]);

while k <> nse+1

do

begin

ntbcrx:=ntbcrx+1;

setntbent;

i:=j+1;

j:=j+nnmmx1;

if k = 1

then

begin

for nnmcx:=1 to nnmmx1

do rtb[rtbcrx].rnn[nnmcx]:=ntb[ntbcrx].nrm[nnmcx];

rtb[rtbcrx].ntbptr:=ntbcrx;

ntb[ntbcrx].fap:=rtbcrx;

fstidx:=ntbcrx

end;

if ntb[ntbcrx].npx=0

then

k:=nse+1

else

k:=k+1

end;

while stk[stkcrx].buf.nxr<>0 do

begin

nxrkpx:=stk[stkcrx].buf.nxr;

rdlin(stk[stkcrx]);

k:=1;

```

i:=1;
j:=nnmmx1;
while k <> nse+1
do
begin
ntbcrx:=ntbcrx+1;
setntbent;
i:=j+1;
j:=j+nnmmx1;
if ntb[ntbcrx].nxp <> 0
then k:=k+1
else k:=nse+1
end;
free(crtfnn,nxrkpx)
end;

ntb[fstidx].rtp:=ntbcrx;
lstidx:=ntbcrx;
for ntbcrx:=fstidx+1 to lstidx
do
begin
ntb[ntbcrx].rtp:=fstidx;
ntb[ntbcrx-1].nxp:=ntbcrx;
ntb[ntbcrx].fap:=ntb[ntbcrx].fap+fstidx-1
end;
ntb[lstidx].nxp:=0;
ntbcrx:=lstidx

end; (*CHGNTB*)


procedure trtint;


procedure chgempftb;


begin
ftbcrx:=ftbcrx+1;
for fplcrx:=bnb+1 to fplmx1
do ftb[ftbcrx][fplcrx]:=true
end; (*CHGEMPFTB*)


procedure chgftb(var crtfnm:fskent);


var tamp:inrec;
i,j,k:integer;
conversion:packed array [1..blnmx1] of boolean;


begin

if crtfnm=fsk[2] then flenum:=1
else if crtfnm=fsk[3] then flenum:=2
else if crtfnm=fsk[4] then flenum:=3
else if crtfnm=fsk[5] then flenum:=4;

```



```

i:=0;
repeat

  readfile(flenum,i,tamp);
  moveleft(tamp.idt,conversion,idtmx1);

  for j:=1 to blnmx1
    do ftb[ftbcrx][i*blnmx1+j-1]:=conversion[j];
  i:=i+1

until i=bnb-1;

k:=((fplmx1+1) mod blnmx1);
readfile(flenum,i,tamp);
moveleft(tamp.idt,conversion,idtmx1);

for j:=1 to k
  do ftb[ftbcrx][(bnb-1)*blnmx1+j-1]:=conversion[j];

rtb[rtbcrx].ftbptr:=ftbcrx

end; (*CHGFTB*)

procedure chgtbs;

begin
  rtbcrx:=rtbcrx+1;
  ftbcrx:=ftbcrx+1;
  chgftb(fsk[fekcrx]);
  chgntb(fsk[fekcrx]);
end; (*CHGTBS*)

procedure setfsk(var crf:fskent);

var crfcrx:0..nnmmx1;

begin
  for crfcrx:=1 to nnmmx1
    do
      begin
        idtcrx:=idtcrx+1;
        crf[crfcrx]:=stk[stkcrx].buf.idt[idtcrx]
      end
    end;
end; (*SETFSK*)

procedure inttrsarg;

begin

  for i:=1 to 4 do trsarg[i,1]:=-2;
  trsarg[0,1]:=1;
  trsarg[0,2]:=-2;
  trsarg[1,2]:=3;

```

```

    trsarg[2,2]:=3;
    trsarg[3,2]:=-2;
    trsarg[4,2]:=-2;
    for i:=0 to 2 do trsarg[i,3]:=-1;
    trsarg[3,3]:=-2;
    trsarg[4,3]:=-1;
    trsarg[0,4]:=-2;
    trsarg[1,4]:=2;
    trsarg[2,4]:=-2;
    trsarg[3,4]:=4;
    trsarg[4,4]:=-2;
    for i:=0 to 4 do trsarg[i,5]:=-2
end; (*INTTRSARG*)

```

procedure intactarg;

begin

```

    actarg[0,1]:=0;
    for i:=1 to 4 do actarg[i,1]:=-1;
    actarg[0,2]:=-1;
    actarg[1,2]:=12;
    actarg[2,2]:=0;
    actarg[3,2]:=-1;
    actarg[4,2]:=-1;
    actarg[0,3]:=3;
    actarg[1,3]:=13;
    actarg[2,3]:=23;
    actarg[3,3]:=-1;
    actarg[4,3]:=0;
    actarg[0,4]:=-1;
    actarg[1,4]:=14;
    actarg[2,4]:=-1;
    actarg[3,4]:=34;
    actarg[4,4]:=-1;
    for i:=0 to 4 do actarg[i,5]:=-1;
end; (*INTACTARG*)

```

procedure intrsnid;

begin

for i:=1 to 4

do

begin

```

    trsnid[i,1]:=-2;
    trsnid[i,2]:=-2;
    trsnid[i,3]:=3;
    trsnid[i,4]:=4;
    trsnid[i,5]:=-2;
    trsnid[i,6]:=-2

```

end;

```

    trsnid[0,1]:=1;
    trsnid[0,2]:=2;
    trsnid[1,2]:=2;
    trsnid[0,3]:=3;
    trsnid[2,3]:=-2;
    trsnid[0,4]:=4;
    trsnid[0,5]:=-2;
    trsnid[4,5]:=-1;
    trsnid[0,6]:=-2;
    trsnid[5,1]:=-2;
    trsnid[5,2]:=-2;

```



```

    trsnid[5,3]:=3;
    trsnid[5,4]:=5;
    trsnid[5,5]:=-2;
    trsnid[5,6]:=-2;
end; (*INTTRSNID*)

```

```

procedure intactnid;

```

```

begin
    for i:=1 to 4
        do actnid[i,1]:=-1;
    actnid[0,1]:=1;
    actnid[0,2]:=0;
    actnid[1,2]:=0;
    actnid[2,2]:=-1;
    actnid[3,2]:=-1;
    actnid[4,2]:=-1;
    actnid[0,3]:=3;
    actnid[1,3]:=3;
    actnid[2,3]:=-1;
    actnid[3,3]:=3;
    actnid[4,3]:=43;
    actnid[0,4]:=4;
    actnid[1,4]:=4;
    actnid[2,4]:=24;
    actnid[3,4]:=4;
    actnid[4,4]:=44;
    for i:=0 to 3
        do actnid[i,5]:=-1;
    actnid[4,5]:=45;
    for i:=0 to 4
        do actnid[i,6]:=-1;
    actnid[5,1]:=-1;
    actnid[5,2]:=-1;
    actnid[5,3]:=43;
    actnid[5,4]:=44;
    actnid[5,5]:=-1;
    actnid[5,6]:=-1;
end; (*INTACTNID*)

```

```

begin

```

```

    writeln('DONNEZ LE NOM DU FICHIER DE COMMANDE(15 CHAR AU MAXIMUM):');
    readln(flenme);
    reset(f0,flenme);

```

```

    lggnbr:=0;
    stkrx:=1;
    askrx:=1;
    ntbrx:=0;
    rtbrx:=0;
    ftbrx:=0;
    askrx:=1;
    rnb:=0;
    crtlev:=0;

```

```

    writeln('DONNEZ LE NOM DE LA DISQUETTE DE L"APPLICATION:');

```

```

    i:=0;
    repeat
        i:=i+1;

```

```

    read(car);
    password[i]:=car
until (i=10) or (car=' ');
if i<10
    then
        repeat
            i:=i+1;
            password[i]:=' ';
        until i=10;

prcfile:=0;
prcblk:=0;
prcop:=false;

verifdsk;
unitread(5,dskbuf,12,0);
fstblk[1]:=dskbuf[1].blk1stptr;

for fskcrx:=1 to fskmx1 do
    begin
        readf0(stk[stkcrx].buf.idt);
        if stk[stkcrx].buf.idt[1] = '*'
            then
                begin
                    idtcrx:=1;
                    setfsk(fsk[fskcrx]);
                    chgtbs;
                    get(f0)
                end
            else
                begin
                    idtcrx:=0;
                    setfsk(fsk[fskcrx]);
                    if fskcrx<>1 then chgempftb;
                    get(f0)
                end;
    end;
ntbfrp:=ntbcrx+1;
for ntbcx:=ntbfrp to ntbm1-1 do
    begin
        ntb[ntbcx].nxp:=ntbcrx+1;
        for nnmcx:=1 to nnmm1 do ntb[ntbcx].nnm[nnmcx]:=' ';
    end;
for nnmcx:=1 to nnmm1
    do ntb[ntbm1].nnm[nnmcx]:=' ';
ntb[ntbm1].nxp:=0;

rtbkpx:=rtbcrx+1;
for rtbcx:=rtbkpx to rtbm1 do
    for nnmcx:=1 to nnmm1 do rtb[rtbcx].nnm[nnmcx]:=' ';

inttrsarg;
intactarg;
inttrenid;
intactnid;

for nnmcx:=1 to nnmm1
    do stk[stkcrx].fnn[nnmcx]:=fsk[1][nnmcx]

end;(*TRTINT*)

```



```
procedure trttr;
```

```
const pntmx1 = 40;
```

```
type pntent = record
```

```
    nnm:packed array[1..nnmmx1] of char;
```

```
    lnb:integer;
```

```
    rtp:0..ntbmx1;
```

```
    nxp:0..ntbmx1;
```

```
    fap:0..ntbmx1;
```

```
    rcp:integer
```

```
end;
```

```
var pnt:array[1..pntmx1] of pntent;
```

```
    crtfnm:packed array[1..nnmmx1] of char;
```

```
    adr:integer;
```

```
    pntcrx,pntkpx:0..pntmx1;
```

```
    crtlev:integer;
```

```
    blanc:packed array[1..nnmmx1] of char;
```

```
procedure fndfpl(var crtadr:integer);
```

```
begin
```

```
    fplcrx:=bnb+1;
```

```
    while (ftb[ftbcrx][fplcrx]=false) and (fplcrx<=fplmx1)
```

```
    do fplcrx:=fplcrx+1;
```

```
    if fplcrx>fplmx1
```

```
    then
```

```
        begin
```

```
            numerr:=8;
```

```
            erreur(numerr)
```

```
        end;
```

```
    ftb[ftbcrx][fplcrx]:=false;
```

```
    crtadr:=fplcrx
```

```
end;(*FNDFPL*)
```

```
procedure wrntnb(var crtadr:integer);
```

```
begin
```

```
    if crtfnm=fsk[2] then flenum:=1
```

```
    else if crtfnm=fsk[3] then flenum:=2
```

```
    else if crtfnm=fsk[4] then flenum:=3
```

```
    else if crtfnm=fsk[5] then flenum:=4;
```

```
    writefile(flenum,crtadr,outrec)
```

```
end;(*WRTNTB*)
```

```
procedure setrec;
```

```
var i, j, k:integer;
```

```
    conversion: packed array [1..2] of char;
```

```
begin
```

```
    k:=1;
```

```
    i:=0;
```

```
    j:=0;
```

```
    while k <> nse + 1 do
```

```

begin
  i:=j+1;
  j:=j+nnmmx1;
  nnmcrcx:=1;
  for odrcrx:=i to j
    do
      begin
        outrec.odt[odrcrx]:=pnt[pntcrx].nnm[nnmcrcx];
        nnmcrcx:=nnmcrcx+1;
      end;

  moveleft(pnt[pntcrx].lnb,conversion,2);

  i:=j+1;
  j:=j+2;
  for odrcrx:=i to j
    do outrec.odt[odrcrx]:=conversion[odrcrx-j+2];

  moveleft(pnt[pntcrx].fap,conversion,2);
  i:=j+1;
  j:=j+2;
  for odrcrx:=i to j
    do outrec.odt[odrcrx]:=conversion[odrcrx-j+2];

  moveleft(pnt[pntcrx].nyp,conversion,2);
  i:=j+1;
  j:=j+2;
  for odrcrx:=i to j
    do outrec.odt[odrcrx]:=conversion[odrcrx-j+2];

  moveleft(pnt[pntcrx].rtp,conversion,2);
  i:=j+1;
  j:=j+2;
  for odrcrx:=i to j
    do outrec.odt[odrcrx]:=conversion[odrcrx-j+2];

  moveleft(pnt[pntcrx].rcp,conversion,2);
  i:=j+1;
  j:=j+2;
  for odrcrx:=i to j
    do outrec.odt[odrcrx]:=conversion[odrcrx-j+2];

  pntcrx:=pnt[pntcrx].nyp;
  if pntcrx <> 0
    then k:=k+1
    else k:=nse+1

  end;

end; (*SETREC*)

```

```

procedure rchftb;

```

```

var i,j,k:integer;
    tamp:rec;
    conversion:packed array [1..blnmx1] of boolean;

```

```

begin
  if crtfnm=fsk[2] then flenum:=1

```



```

else if crtfnm=fsk[3] then flenum:=2
else if crtfnm=fsk[4] then flenum:=3
else if crtfnm=fsk[5] then flenum:=4;
i:=0;

repeat
  for j:=1 to blnmxl
    do conversion[j]:=ftb[ftbcrx][i*blnmxl+j-1];
    moveleft(conversion,tamp.odt,odtmxl);
    writefile(flenum,i,tamp);
    i:=i+1
  until i = bnb-1;

  k:=((fplmxl+1) mod blnmxl);
  for j:=1 to k
    do conversion[j]:=ftb[ftbcrx][i*blnmxl+j-1];
    moveleft(conversion,tamp.odt,odtmxl);
    writefile(flenum,i,tamp)

end; (*RCHFTB*)

```

procedure rchntb;

```

begin
  pntcrx:=1;
  crtnxr:=bnb;
  repeat
    setrec;
    if pntcrx = 0
      then
        begin
          outrec.nxr:=0;
          wrtnb(crtnxr)
        end
      else
        begin
          fndfpl(outrec.nxr);
          wrtnb(crtnxr)
        end;
    crtnxr:=outrec.nxr
  until pntcrx = 0
end; (*RCHNTB*)

```

procedure updfap;

var pntidx:0..pntmxl;

```

begin
  pntcrx:=0;
  repeat
    pntkpx:=pntcrx;
    pntcrx:=pntcrx+1;
    if htb[ntbcrx].lnb = 0
      then
        begin
          for i:=1 to nmmxl
            do pnt[pntcrx].nmm[i]:=ntb[ntbcrx].nmm[i];

```

```

    pnt[pntcrx].lnb:=ntb[ntbcrx].lnb;
    if ntb[ntbcrx].nxp = 0
    then pnt[pntcrx].nxp:=0
    else pnt[pntcrx].nxp:=pntcrx+1;
    pnt[pntcrx].rcp:=ntb[ntbcrx].rcp;
end
else
begin
    crtlev:=pnt[pntkpx].lnb;
    for i:=1 to nmmx1
    do pnt[pntcrx].nmm[i]:=ntb[ntbcrx].nmm[i];
    pnt[pntcrx].lnb:=ntb[ntbcrx].lnb;
    pnt[pntcrx].rtp:=1;
    if ntb[ntbcrx].nxp = 0
    then pnt[pntcrx].nxp:=0
    else pnt[pntcrx].nxp:=pntcrx+1;
    pnt[pntcrx].rcp:=ntb[ntbcrx].rcp;
    if pnt[pntcrx].lnb = crtlev
    then pnt[pntcrx].fap:=pnt[pntkpx].fap;
    if pnt[pntcrx].lnb = crtlev +1
    then pnt[pntcrx].fap:=pntkpx;
    if pnt[pntcrx].lnb < crtlev
    then
        begin
            pntidx:=pntkpx;
            while pnt[pntidx].lnb > pnt[pntcrx].lnb
            do pntidx:=pnt[pntidx].fap;
            pnt[pntcrx].fap:=pnt[pntidx].fap
        end
    end;

    ntbcrx:=ntb[ntbcrx].nxp
until ntbcrx = 0;
pnt[1].rtp:=pntcrx
end; (*UPDFAP*)

begin
    rtbcrx:=1;
    for i:=1 to nmmx1
    do blanc[i]:=' ';
    while rtbcrx <= rtbmx1
    do
        begin
            if rtb[rtbcrx].rnm = blanc
            then rtbcrx:=rtbcrx+1
            else
                begin
                    ntbcrx:=rtb[rtbcrx].ntbptr;
                    ftbcrx:=rtb[rtbcrx].ftbptr;
                    for nmmcrx:=1 to nmmx1 do
                        crtfrm[nmmcrx]:=rtb[rtbcrx].rnm[nmmcrx];

                    updfap;
                    rchntb;
                    rchftb;
                    rtbcrx:=rtbcrx+1
                end
            end;
        end
    rchblktbl;
    close(fO)
end; (*TRTTER*)

begin
end.
```


(**S+*)

unit contenu;

interface

uses (**U APPLE2:DECLIB.CODE *)decl,
(**U APPLE2:INTERLIB.CODE *)inter;

procedure trtgen(var crtadr:integer);
procedure trtendcre(var crtadr:integer);
procedure setcrn;
procedure trtconnod(var crtadr:integer);
procedure trtconval(var crtadr:integer);

implementation

procedure trtgen;

var fstnbr,secnbr:integer;

procedure trtarg;

var ste,oldste,code:integer;

function codarg:integer;

begin

if (crtcar<>'W') and (crtcar<>',') and (crtcar<>'(') and not (crtcar='0')
and not (crtcar='1')
and not (crtcar='2')
and not (crtcar='3')
and not (crtcar='4')
and not (crtcar='5')
and not (crtcar='6')
and not (crtcar='7')
and not (crtcar='8')
and not (crtcar='9')

then codarg:=5

else

case crtcar of
'W':codarg:=1;
,':codarg:=2;
'(':codarg:=3;
'0','1','2','3','4','5','6','7','8','9':codarg:=4

end

end; (**CODARG*)

```

function valnbr:integer;
var oldvalnbr,nbr:integer;

begin
  oldvalnbr:=0;
  moveright(crtcar,oldvalnbr,1);
  oldvalnbr:=oldvalnbr-48;
  getsgncar(stk[stkcrx]);
  while (crtcar = '0') or
        (crtcar = '1') or
        (crtcar = '2') or
        (crtcar = '3') or
        (crtcar = '4') or
        (crtcar = '5') or
        (crtcar = '6') or
        (crtcar = '7') or
        (crtcar = '8') or
        (crtcar = '9')
  do
    begin
      nbr:=0;
      moveright(crtcar,nbr,1);
      nbr:=nbr-48;
      oldvalnbr:=oldvalnbr*10+nbr;
      getsgncar(stk[stkcrx])
    end;
    valnbr:=oldvalnbr
  end; (*VALNBR*)

```

```

procedure absarg;

```

```

begin
  fstnbr:=0;
  secnbr:=0;
  getsgncar(stk[stkcrx])
end; (*ABSARG*)

```

```

procedure absfstsecnbr;

```

```

begin
  fstnbr:=0;
  secnbr:=maxint;
  getsgncar(stk[stkcrx])
end; (*ABSFSTSECNBR*)

```

```

procedure absfstnbr;

```

```

begin
  fstnbr:=0;
  getsgncar(stk[stkcrx])
end; (*ABSFSTNBR*)

```


procedure abssecnbr;

begin
 secnbr:=maxint;
 getsgncar(stk[stkcrx])
end; (*ABSSECNBR*)

procedure trtfstnbr;

begin
 fstnbr:=valnbr
end; (*TRTFSTNBR*)

procedure trtsecnbr;

begin
 secnbr:=valnbr
end; (*TRTSECNBR*)

begin
 ste:=0;
 while ste >= 0
 do
 begin
 oldste:=ste;
 code:=codarg;
 ste:=trsarg[oldste,code];
 case actarg[oldste,code] of -1:begin
 numerr:=1;
 erreur(numerr)
 end;
 13:abefstsecnbr;
 3:absarg;
 23:abssecnbr;
 14:trtfstnbr;
 34:trtsecnbr;
 12:abfstnbr;
 0:getsgncar(stk[stkcrx])
 end
 end
end; (*TRTARG*)

procedure genrec(var crtadr:integer);

begin
 stk[stkcrx].crp:=idtcx;
 stkcrx:=stkcrx+1;
 if ntb[ntbcrx].rcp <> 0
 then
 begin
 stk[stkcrx].buf.nxr:=ntb[ntbcrx].rcp;
 if ntb[ntbcrx].lnb<>0
 then stk[stkcrx].fnm:=ntb[ntb[ntbcrx].rtp].nnm
 else stk[stkcrx].fnm:=ntb[ntbcrx].nnm;
 end
 repeat
 rdlin(stk[stkcrx]);

```

while idtcx <> idtmx1
do
begin
getcar(stk[stkcrx]);
if crtcar <> '/'
then putcar(crtadr)
else
begin
getcar(stk[stkcrx]);
if crtcar = 'G'
then
begin
getcar(stk[stkcrx]);
ssk[sskcrx].ntbcrx:=ntbcrx;
ssk[sskcrx].askkpx:=askkpx;
sskcrx:=sskcrx+1;
trtgen(crtadr);
sskcrx:=sskcrx-1;
ntbcrx:=ssk[sskcrx].ntbcrx;
askkpx:=ssk[sskcrx].askkpx;
end
end
else
begin
numerr:=1;
erreur(numerr)
end
end
end
until stk[stkcrx].buf.nxr = 0
end;
stkcrx:=stkcrx-1;
idtcx:=stk[stkcrx].crp
end; (*GENREC*)

```

```

procedure gen(var crtadr:integer);

```

```

var oldlev:integer;
    bool:boolean;

```

```

begin
repeat
askcrx:=askcrx-1;
ntbcrx:=ask[askcrx];
oldlev:=ntb[ntbcrx].lnb;
if (fstnbr = 0) and (secnbr = 0)
then genrec(crtadr)
else
repeat
if (secnbr=maxint) or ((secnbr<maxint) and
(ntb[ntbcrx].lnb <= oldlev+secnbr))
then
if ntb[ntbcrx].lnb >= oldlev+fstnbr
then genrec(crtadr);
ntbcrx:=ntb[ntbcrx].nxp;
if ntbcrx=0
then bool:=true
else if ntb[ntbcrx].lnb=oldlev
then bool:=true
else bool:=false
until bool
until askcrx = askkpx

```



```
end; (*GEN*)
```

```
begin
  trtarg;
  trtnid;
  if (mny = false) and (nnb > 1)
  then
    begin
      numerr:=6;
      erreur(numerr)
    end
  else
    if nnb = 0
    then
      begin
        numerr:=5;
        erreur(numerr)
      end
    else gen(crtadr)
  end; (*TRTGEN*)
```

```
procedure trtendcre;
```

```
var crtfnm:packed array[1..nnmmx1] of char;
    flenum,i:integer;
```

```
begin
  rnb:=rnb+1;
  if rnb=1 then begin
    schfpl(crtadr);
    ntb[crtadr].rcp:=fplkpx;
    crtnxr:=fplkpx
  end;
  outrec.nxr:=0;
  for i:=1 to nnmmx1
  do crtfnm[i]:=ntb[ntb[crtadr].rtp].nnm[i];
  if crtfnm=fsk[2] then flenum:=1;
  if crtfnm=fsk[3] then flenum:=2;
  if crtfnm=fsk[4] then flenum:=3;
  if crtfnm=fsk[5] then flenum:=4;

  if odterx<odtmx1
  then
    for i:=odterx+1 to odtmx1
    do outrec.odt[i]:=' ';

  writefile(flenum,crtnxr,outrec);

  rnb:=0

end; (*TRTENDCRE*)
```

```

procedure setcrn;
begin
  for crncrx:=1 to nmmx1 do crn[crncrx]:= ' ';
  crncrx:=0;
  while ((crtcar <> ' ') and (crtcar <> ';')) and (crncrx < nmmx1)
    do begin
      crncrx:=crncrx+1;
      crn[crncrx]:=crtcar;
      getsncar(stk[stkcrx])
    end;
  if (crtcar <> ' ') and (crtcar <> ';')
    then begin
      numerr:=1;
      erreur(numerr)
    end
end; (*SETCRN*)

```

```

procedure trtconnod;
var abscar:char;
begin
  abscar:= ' ';
  if crtcar <> ' ' then begin
    numerr:=1;
    erreur(numerr)
  end;

  repeat
    getcar(stk[stkcrx]);
    if crtcar = ' ' then begin
      getcar(stk[stkcrx]);
      if crtcar <> ' '
        then crtcar:=chr(3)
      end;

      abscar:=crtcar;
      putcar(crtadr)
    until abscar=chr(3);
    trtendcre(crtadr)
  end; (*TRTCONNOD*)

```

```

procedure trtconval;
var abscar:char;
begin
  rcslev:=0;

  if crtcar <> ' ' then
    begin
      numerr:=1;
      erreur(numerr)
    end;

  abscar:= ' ';

  repeat
    getcar(stk[stkcrx]);

```



```

if crtcar = '/'
then begin
    getsgncar(stk[stkcrx]);
    if crtcar = '0'
    then begin
        getsgncar(stk[stkcrx]);
        trtgen(crtadr);
        getcar(stk[stkcrx])
    end
    else begin
        numerr:=1;
        erreur(numerr)
    end
end;
if crtcar = ""
then
begin
    getcar(stk[stkcrx]);
    if crtcar <> ""
    then crtcar:=chr(3)
end;
abscar:=crtcar;
putcar(crtadr)
until abscar = chr(3);
trtendcre(crtadr)
end; (*TRTCONVAL*)

begin
end.

```

(*S+*)

program test(input,output);

(*U APPLE2:DECLIB.CODE *)

uses decl;

var trouve:boolean;
 reponse,choix:char;
 outfile:interactive;
 device:string[8];
 j:integer;

procedure affcon;

begin

 write(outfile,'FICHIER: ');
 writeln(outfile,stk[stkcrx].fnn);
 writeln(outfile);

 repeat

 write(outfile,'NNM: ',ntb[ntbcrx].nnm);
 write(outfile,' LNB: ',ntb[ntbcrx].lnb,' FAP: ',ntb[ntbcrx].fap);
 write(outfile,' NXP: ',ntb[ntbcrx].nxp,' RTP: ',ntb[ntbcrx].rtp);
 write(outfile,' RCP: ',ntb[ntbcrx].rcp);
 writeln(outfile);

 if ntb[ntbcrx].rcp<>0

 then

 begin

 stk[stkcrx].buf.nxr:=ntb[ntbcrx].rcp;
 repeat
 writeln(stk[stkcrx].buf.nxr);
 readfile(flenum,stk[stkcrx].buf.nxr,stk[stkcrx].buf);
 writeln(outfile,stk[stkcrx].buf.idt);
 until stk[stkcrx].buf.nxr=0;
 writeln(outfile)

 end;

 ntbcrx:=ntb[ntbcrx].nxp

 until ntbcrx=0;

end; (*RESULT*)

procedure afftab;

begin

 writeln(outfile,'TABLE DES NOEUDS DU FICHIER ',stk[stkcrx].fnn);
 writeln(outfile,'=====');
 writeln(outfile);

 writeln(outfile,'*****');
 writeln(outfile,'* NTCRX * NNM * LNB * FAP * NXP * RTP * RCP *');
 writeln(outfile,'*****');

 repeat

 writeln(outfile,'* ',ntbcrx:4,' * ',ntb[ntbcrx].nnm,' * ',ntb[ntbcrx].lnb:4,
 ' * ',ntb[ntbcrx].fap:4,' * ',ntb[ntbcrx].nxp:4,' * ',ntb[ntbcrx].rtp:4,
 ' * ',ntb[ntbcrx].rcp:4,' * ');


```

ntbcrx:=ntb[ntbcrx].nxp
until ntbcrx=0;
writeln(outfile,'*****')
end(*AFFTAB*);

```

```

procedure chgntb(var crtfnm:fsknt);

```

```

var i,j,k:integer;
fstidx,lstidx:0..ntbm-1;

```

```

procedure setntbent;

```

```

var conversion:packed array[1..2] of char;

```

```

begin

```

```

  nmcrcx:=1;
  for idtcrx:=1 to j
    do
      begin
        ntb[ntbcrx].nmc[nmcrcx]:=stk[stkcrx].buf.idt[idtcrx];
        nmcrcx:=nmcrcx+1;

```

```

      end;

```

```

  i:=j+1;
  j:=j+2;
  for idtcrx:=i to j
    do conversion[idtcrx-j+2]:=stk[stkcrx].buf.idt[idtcrx];
  moveleft(conversion,ntb[ntbcrx].lnb,2);

```

```

  i:=j+1;
  j:=j+2;
  for idtcrx:=i to j
    do conversion[idtcrx-j+2]:=stk[stkcrx].buf.idt[idtcrx];
  moveleft(conversion,ntb[ntbcrx].fap,2);

```

```

  i:=j+1;
  j:=j+2;
  for idtcrx:=i to j
    do conversion[idtcrx-j+2]:=stk[stkcrx].buf.idt[idtcrx];
  moveleft(conversion,ntb[ntbcrx].nxc,2);

```

```

  i:=j+1;
  j:=j+2;
  for idtcrx:=i to j
    do conversion[idtcrx-j+2]:=stk[stkcrx].buf.idt[idtcrx];
  moveleft(conversion,ntb[ntbcrx].rtp,2);

```

```

  i:=j+1;
  j:=j+2;
  for idtcrx:=i to j
    do conversion[idtcrx-j+2]:=stk[stkcrx].buf.idt[idtcrx];
  moveleft(conversion,ntb[ntbcrx].rcp,2);

```

```

end(*SETNTBENT*);

```

```

begin

```

```

k:=1;
i:=1;
j:=nnmmx1;

for nnmcx:=1 to nnmmx1
  do stk[stkcrx].fcm[nnmcx]:=crtfcm[nnmcx];
  stk[stkcrx].buf.nxr:=bnb;
  rdlin(stk[stkcrx]);
  while k <> nse+1
    do
      begin
        ntbcx:=ntbcx+1;

        setntbent;

        i:=j+1;
        j:=j+nnmmx1;
        if k = 1
          then
            begin
              for nnmcx:=1 to nnmmx1
                do rtb[rtbcx].rnm[nnmcx]:=ntb[ntbcx].rnm[nnmcx];
                rtb[rtbcx].ntbptr:=ntbcx;
                ntb[ntbcx].fap:=rtbcx;
                fstidx:=ntbcx
              end;

              if ntb[ntbcx].nxp=0
                then
                  k:=nse+1
                else
                  k:=k+1
            end;

while stk[stkcrx].buf.nxr<>0 do

begin

  rdlin(stk[stkcrx]);
  k:=1;
  i:=1;
  j:=nnmmx1;
  while k <> nse+1
    do
      begin
        ntbcx:=ntbcx+1;
        setntbent;
        i:=j+1;
        j:=j+nnmmx1;
        if ntb[ntbcx].nxp <> 0
          then k:=k+1
          else k:=nse+1
        end
      end;

ntb[fstidx].rtp:=ntbcx;
lstidx:=ntbcx;
for ntbcx:=fstidx+1 to lstidx
  do
    begin
      ntb[ntbcx].rtp:=fstidx;
      ntb[ntbcx-1].nxp:=ntbcx;

```



```

        ntb[ntbcrx].fap:=ntb[ntbcrx].fap+fstidx-1
    end;
    ntb[lstidx].nxp:=0;

```

```

end; (*CHGNTB*)

```

```

begin

```

```

    writeln('SUR QUEL DEVICE? CONSOLE: OU PRINTER:');
    readln(device);
    reset(outfile,device);

```

```

    writeln('NOM DE LA DISQUETTE');

```

```

    i:=0;

```

```

    repeat

```

```

        i:=i+1;

```

```

        read(car);

```

```

        password[i]:=car

```

```

    until (i=10) or (car=' ');

```

```

    if i<10

```

```

    then

```

```

        repeat

```

```

            i:=i+1;

```

```

            password[i]:=' ';

```

```

        until i=10;

```

```

    writeln;

```

```

    prcfile:=0;

```

```

    prcblk:=0;

```

```

    prcop:=false;

```

```

    writeln('FICHIERS DE L'APPLICATION: (MAX NNNMXXL CHAR)');

```

```

    for j:=2 to 5

```

```

    do

```

```

        begin

```

```

            write('FICHIER ',j,' : ');

```

```

            i:=0;

```

```

            repeat

```

```

                i:=i+1;

```

```

                read(car);

```

```

                fsk[j][i]:=car

```

```

            until (i=nnmmxl) or (car=' ');

```

```

            if i<nnmmxl

```

```

            then

```

```

                repeat

```

```

                    i:=i+1;

```

```

                    fsk[j][i]:=' ';

```

```

                until i=nnmmxl;

```

```

            writeln

```

```

        end;

```

```

    for i:=1 to nnmmxl

```

```

    do fsk[i][i]:=' ';

```

```

    skcrx:=1;

```

```

    repeat

```

```

        writeln('NUM DU FICHIER A TESTER: (DE 2 A 5)');

```

```

        readln(fskcrx);

```

```

    repeat

```

```

        writeln('TABLE? (-->T), CONTENU? (-->C), LES DEUX? (-->D)');

```

```

        readln(choix)

```

```
until (choix='C') or (choix='T') or (choix='D');
```

```
ntbcrx:=0;
```

```
rtbcrx:=1;
```

```
ftbcrx:=1;
```

```
flenum:=fskcrx-1;
```

```
chgntb(fsk[fskcrx]);
```

```
trouve:=false;
```

```
rtbcrx:=0;
```

```
repeat
```

```
  rtbcrx:=rtbcrx+1;
```

```
  if fsk[fskcrx]=rtb[rtbcrx].rnm
```

```
    then
```

```
      begin
```

```
        ntbcrx:=rtb[rtbcrx].ntbptr;
```

```
        trouve:=true
```

```
      end
```

```
until trouve or (rtbcrx=rtbmxl);
```

```
if not trouve
```

```
  then
```

```
    begin
```

```
      writeln('ERREUR SUR LE NOM DU FICHIER.');
```

```
      writeln('IL NE CORRESPOND PAS A UN NOM DE RACINE.');
```

```
      exit(program)
```

```
    end;
```

```
case choix of 'C':affcon;
```

```
              'T':afftab;
```

```
              'D':begin
```

```
                  afftab;
```

```
                  affcon
```

```
                end
```

```
end;
```

```
writeln('<RETURN> POUR CONTINUER, 0<RETURN> POUR ARRETER');
```

```
readln(reponse)
```

```
until reponse='0'
```

```
end.(*TEST*)
```


ANNEXE DEUX

LE MANUEL UTILISATEUR

1. Configuration physique. Disquettes

Le système de gestion de fichiers a été implémenté sur un micro-ordinateur Apple II de 16 K octets, relié à une console et à deux disk drive.

Trois disquettes ont été utilisées:

i) Apple 1 qui contient

une partie du système d'exploitation,
les versions text et code du programme TEST,
une partie de la version text du programme
SGFA
la version code du programme SGFA

ii) Apple 2 qui contient

une partie du système d'exploitation,
les bibliothèques DECL, INTER et CONT, dans les
fichiers DECLIB, INTERLIB et CONTLIB,
la suite du programme SGFA, dans le fichier
CIMAD.TEXT
(cette suite est compilée dans SGFA grâce
à l'option include file du compilateur)

iii) On a aussi une disquette vierge destinée à servir
de support aux fichiers de l'application.

2. Initialisation de la disquette vierge.

Cette procédure est dangereuse, car elle écrit dans les blocs de la disquette qui se trouve dans le drive 2. Avant d'en lancer l'exécution, vérifier que la bonne disquette se trouve bien dans le drive!

Avec la disquette vierge dans le drive 2, lancer l'exécution du programme d'initialisation qui se trouve dans le fichier INIDSK sur Apple 1. Le programme demande d'abord à l'utilisateur s'il veut modifier le nom de la disquette. Ce nom servira d'identification de la disquette, et permettra au système de vérifier que la bonne disquette est en place.

Si l'utilisateur répond "oui", il devra donner une chaîne de 10 caractères. Cette chaîne sera copiée dans le bloc d'adresse 0 de la disquette.

Ensuite, on demande à l'utilisateur s'il désire modifier les pointeurs de la disquette. Ces pointeurs sont ceux qui servent à chaîner entre eux les blocs libres de la disquette. L'initialisation consiste à mettre tous les blocs dans la chaîne, à l'exception des cinq premiers, qui sont réservés à un autre usage. Le pointeur de début de chaîne sera mémorisé dans le bloc d'adresse 0.

Les blocs d'adresse 1 à 4 sont réservés pour y stocker les tables des blocs des 4 fichiers de l'application. Ces tables sont initialisées de telle sorte que les fichiers soient vides, c'est-à-dire avec toutes leurs entrées nulles.

Remarque

On peut changer le nom de la disquette sans perdre son contenu en réinitialisant les pointeurs et les tables.

3. Création du fichier de commande.

Ce fichier est créé à l'éditeur et doit être sauvegardé après sa création pour constituer un fichier de type ".TEXT".

La première ligne du fichier est réservée au nom du fichier de commande. Ce nom n'a aucun rapport avec celui du fichier créé à l'éditeur.

Les quatre lignes suivantes sont réservées aux noms des quatre fichiers de l'application, précédés ou non d'une étoile suivant qu'ils existent déjà ou que ce sont de nouveaux fichiers.

Les lignes suivantes sont les lignes de commande proprement dites. Rappelons en leur syntaxe:

```

CREATE:      /C <fonction>, { (nom de fichier) }01
              <nom de noeud>; { "<contenu>" }01

GENERATE:    /G <argument> ( { ! {01 identifiant de noeud }01 );

END:         /E { <nom de noeud> }01

INSERT:      /I <fonction> ( { ! {01 <identifiant de noeud> }01 );
              <lien de parenté>, <nom de noeud> { "<contenu>" }01

DELETE:      /D ( { ! {01 <identifiant de noeud> }01 );

ADD:         /A ( { ! {01 <identifiant de noeud> }01 ); "<contenu>"

MODIFY:      /M ( { ! {01 <identifiant de noeud> }01 ); <modification>

```

$\langle \text{nom de fichier} \rangle ::= \langle \text{nom de noeud} \rangle$

$$\langle \text{fonction} \rangle ::= \langle V \rangle / \langle R \rangle$$
$$\langle \text{nom de noeud} \rangle ::= \langle \text{lettre} \rangle \{ \langle \text{lettre ou chiffre} \rangle \}$$
$$\langle \text{lettre ou chiffre} \rangle ::= \langle \text{lettre} \rangle / \langle \text{chiffre} \rangle$$

$\langle \text{identifiant de noeud} \rangle ::= \langle \text{nom de racine} \rangle \{ \langle \text{queue d'identifiant} \rangle \}$
 $\langle \text{queue d'identifiant} \rangle ::= \langle \text{séparateur} \rangle \langle \text{nom de noeud} \rangle /$
 $\langle \text{queue d'identifiant} \rangle \langle \text{séparateur} \rangle$
 $\langle \text{nom de noeud} \rangle$
 $\langle \text{séparateur} \rangle ::= * \{ \cdot \} / \cdot \{ \cdot \}$
 $\text{lien de parenté} ::= \langle \text{fils aîné} \rangle / \langle \text{fils cadet} \rangle / \langle \text{frère cadet} \rangle$
 $\langle \text{fils aîné} \rangle ::= \text{EL}$
 $\langle \text{fils cadet} \rangle ::= \text{YO}$
 $\langle \text{frère cadet} \rangle ::= \text{YB}$
 $\langle \text{argument} \rangle ::= \{ W \} \{ \text{entier} \}' \{ \text{entier} \}' \{ \text{entier} \}'$
 $\langle \text{entier} \rangle ::= \langle \text{chiffre} \rangle / \langle \text{chiffre} \rangle \langle \text{entier} \rangle$
 $\langle \text{modification} \rangle ::= \langle \text{nom de noeud} \rangle / " \langle \text{contenu} \rangle "$
 $\langle \text{nom de noeud} \rangle " \langle \text{contenu} \rangle "$
 $\langle \text{contenu} \rangle ::=$ texte structuré en lignes à l'aide du caractère 'carriage return' qui indique la fin d'une ligne.

Le texte peut contenir des commandes
 GENERATE.

4. Exécution du programme d'application SGFA.

La version exécutable de SGFA s'obtient en linkant les bibliothèques DECLIB, INTERLIB et CONTLIB se trouvant sur Apple 2 avec la version compilée de SGFA se trouvant sur Apple 1.

Pour exécuter SGFA, il faut
dans le drive 1, la version exécutable de SGFA,
le fichier des commandes
(sur la disquette Apple 1)
dans le drive 2, la disquette support des fichiers
convenablement initialisée

Au début de l'exécution, on demande le nom du fichier de commande. Il faut donner le nom complet, avec le suffixe .TEXT.

On demande ensuite le nom de la disquette de support. Le programme ne va pas plus loin tant que le nom lu sur la disquette ne correspond pas à celui lu à l'écran.

Une fois la bonne disquette dans le drive, le programme tourne sans intervention de l'utilisateur. Il affiche à l'écran le numéro de la ligne de commande en traitement. Il s'arrête en cas d'erreur, après envoi d'un message explicatif.

5. Exécution du programme TEST.

Ce programme permet de consulter les données des fichiers de l'application: table des noeuds ou contenu des noeuds.

Ce programme se trouve sur Apple 1 qui doit se trouver dans le drive 1. Le drive 2 doit contenir la disquette support des fichiers.

On demande à l'utilisateur

- le nom de la disquette support, pour vérification,
- le nom des fichiers,
- le device choisi pour l'impression: CONSOLE: ou PRINTER:,
- le fichier dont on veut consulter le contenu,
- le type de donnée à tester:
 - la table des noeuds,
 - les contenus des noeuds,
 - les deux.

C O N C L U S I O N S

Nous sommes maintenant arrivés à un point du développement du projet où nous disposons d'une application qui fonctionne. Mais ce fonctionnement n'est pas parfait: nous avons vu que certaines erreurs n'avaient pas été corrigées du fait de leur complexité. d'autre part, la gestion des erreurs n'a pas été développée au delà du message à l'utilisateur et de l'arrêt pur et simple de l'exécution. Cet arrêt pouvant d'ailleurs être la cause d'un grand désagrément, en laissant les données des fichiers dans un état incohérent qui peut parfois obliger l'utilisateur à recréer les fichiers.

On pourrait palier à cela en rechargeant les tables et en s'assurant que le système reste dans un état cohérent. Les données ne seraient peut-être plus correctes mais l'utilisateur, y ayant accès sans problème, pourrait facilement les corriger.

Un autre problème concerne la place sur la disquette qui risque de manquer si les fichiers ont une taille importante. Il peut en effet arriver, suite aux opérations effectuées sur les contenus, qu'un fichier occupe plus de blocs physiques qu'il n'en a réellement besoin, à cause d'une mauvaise répartition des enregistrements.

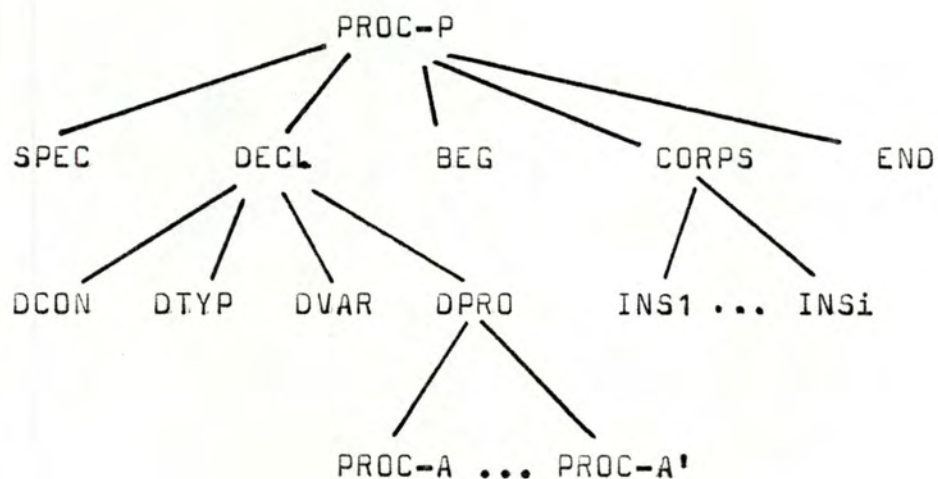
Il faut noter que ce problème ne se pose que pour les enregistrements contenus de noeuds. Les enregistrements de la table des noeuds sont "tassés" à chaque rechargement dans le fichier.

On pourrait prévoir un programme de récupération de place qui tasse les enregistrements occupés au début du fichier et libère ainsi un certain nombre de blocs.

Nous voudrions terminer en donnant un exemple d'utilisation du système que nous venons d'implémenter.

Nous avons dit dans l'introduction que notre objectif était de fournir un outil d'aide à la documentation d'une application informatique. Peut-être pourrions-nous aussi nous servir de cet outil pour gérer tout le texte d'un programme.

Essayons par exemple de décrire une procédure Pascal sous forme arborescente. La structure de tout un programme peut alors être obtenue par récursivité.

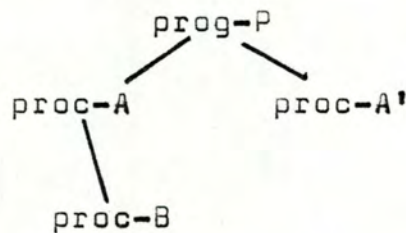


où le contenu de PROC-P sera PROC-P

SPEC	les spécifications de PROC-P
DECL	/
BEG	begin
CORPS	/
END	end
DCON	l'ensemble des déclarations de constantes

DTYP	l'ensemble des déclarations de type
DVAR	l'ensemble des déclarations de variable
DPRO	/
INSi	un groupe d'instructions

Quelles opérations pourrait-on effectuer?
Supposons que nous ayons la structure déclarative suivante:



Comment créer un noeud de spécification qui contiennent les noms de toutes les procédures déclarées dans prog-P?

```

/I V (prog-P.spec); el, spnme "Procédures utilisées:
                               /G 1,1 (*DPRO)"
  
```

On crée ainsi un fils aîné de SPEC, et on y génère les contenus des fils de tous les noeuds DPRO

On peut effectuer le même genre d'opération sur des spécifications ou sur des groupes d'instructions.

```

/I V (prog-P...proc-A'.CORPS.INSi); YB, INSi+1
    "/G (prog-P...proc-A...proc-B.CORPS.INSj)"
  
```

On crée ainsi un noeud frère cadet de INSi dans A' dont le contenu est celui de INSj dans B.

Pour avoir tout le texte du programme, on peut créer un noeud où seront recopiés les contenus de tous les noeuds de l'arbre:

```
/C V,(prgm)PRGM "/G W(prog-P);"
```